第 2 章

Scilab の基礎

本章では Scilab を用いて，手計算だけでは難しい複雑な線型計算を行うための準備訓練を行う。この後の章で連立一次方程式を解く手法と固有値問題を解く手法を扱うが，そのためには Scilab の機能の概要を知らねばならない。本章ではこのための必要最低限の機能について大雑把に解説する。極力，ここに提示した実行例は自分の手で入力し，何が実行され，どういう結果が出たのか，そして，その結果の正しさをどう確認するのかを理解して頂きたい。

2.1 Scilab とは？

Scilab[4] はフランスの国立研究所である INRIA が主導して構築した統合型数値計算ソフトウェアである。C/C++, Fortran のようなコンパイラ言語からは複雑な呼び出し方が必要となる LAPACK のような数値計算ライブラリを，使いやすいスクリプト言語を介して簡単に呼び出すことができるようにしたもので，その源流は Matlab にある。商用サポートなしの Scilab はフリーで使用でき，最新版は下記の Web ページからダウンロードできる。

`http://www.scilab.org/`

2016 年 4 月現在の最新バージョンは 5.5.2 で，Windows 10/8.x や，MacOS X, Linux 上で使用できる。

Scilab 同様の統合型数値計算ツールは Octave 等がある。Matlab は高価な商用ソフトウェアではあるが，歴史的には最も古く，データの可視化機能や付属ツール (Matlab Toolbox) の蓄積では最も先端を走っている。他にも，記号処理機能が優れている数式処理ソフト (Mathematica 等）があり，こちらも数値計算の機能はどんどんレベルアップしている。詳細は各種ソフトウェアの Web ページを参照されたい。

2.2 Scilab の基本演算機能

Scilab にはコマンドライン版と GUI 版が用意されているが，本章では図 2.2 に示す Windows 用の GUI 版に基づいて解説する。

GUI 版は図 2.2 に示すように次の 4 つのウィンドウから構成される。

1. ファイルブラウザ・・・Scilab スクリプト (.sce という拡張子を持つ Scilab 命令から成るプログラ

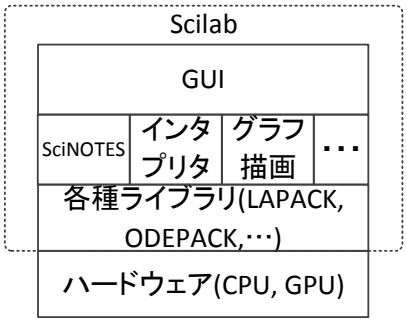

図 2.1　Scilab のソフトウェア構造

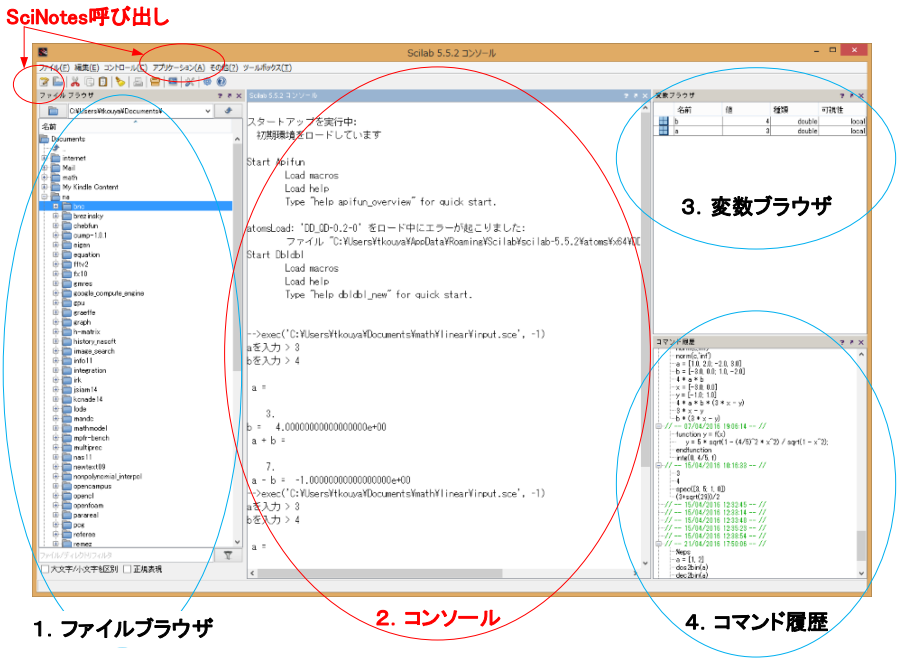

図 2.2　Scilab の起動画面：Windows 用 GUI 版の場合

ム) 等を呼び出す。

2. コンソール・・・Scilab 命令（コマンド）を直接打ち込むことができ，標準入出力はこのコンソール上で行われる。
3. 変数ブラウザ・・・定義済みの Scilab 変数一覧。
4. コマンド履歴・・・コンソールに打ち込んだ Scilab 命令を表示。特定の命令をダブルクリックすることで呼び出すことができる

ここではフリー版の Scilab の基本機能を見ていくことにする。使用方法としては，Scilab を起動したのち，

- コンソールに直接コマンドを打ち込んで 1 行ずつ実行する
- Scilab 付属のテキストエディタ (SciNotes) を起動して実行したいコマンドを連ねたプログラム（Scilab スクリプト) を作成してまとめて実行する

という二つのやり方がある。簡単な実行例はコンソールに直接打ち込み，長い Scilab スクリプトは SciNotes で実行，というように適宜使い分けるようにするとよい。ここではそのように使い分けを行う。

Scilab が行う計算は，特に指定しない限り，全て IEEE754 倍精度 (8 Byte, 2 進 53bits, 10 進約 15 桁) の実数型で行われる。データを記憶するための変数 (variable) は，特に型宣言することなく，スクリプトの任意の位置で使用できる。大文字小文字の区別はあるので，Vec と vec は異なる変数として扱われる。

```
-->Vec = 3 ←ここを打ち込む
 Vec  = ← 変数 Vec に
        ←
    3.  ← 3 が入っているという意味

-->vec = 2 ←ここを打ち込む
 vec  =

    2.
```

一度使用した変数は clear 命令を使って消去しない限り記憶し続ける。

```
-->Vec        ← 変数 Vec に格納されているデータを表示
 Vec  =

       3.

-->clear Vec ← 変数 Vec を消去

-->Vec
       !--error 4
変数は定義されていません: Vec ← 変数 Vec が消えているのでエラーとなる
```

変数の値をコンソールに出力したくないときにはセミコロン; を使用する。

```
-->Vec = 3 ← 変数 Vec に 3 を格納し，コンソールに Vec の値を出力
 Vec  =

    3.

-->Vec; ← Vec の値を表示しない
```

```
-->Vec   ← Vec の値を表示
 Vec  =

    3.
```

変数に値が入力されたら，それを用いて演算ができる。四則演算は下記のように実行する。式に含まれる半角スペースはあってもなくてもよいが，見易さのため，必要に応じて入れる癖をつけておくとよい。

```
-->Vec + vec ← Vec と vec の和
 ans  =

    5.

-->Vec - vec ← Vec と vec の差
 ans  =

    1.

-->Vec * vec ← Vec と vec の積
 ans  =

    6.

-->Vec / vec ← Vec と vec の商
 ans  =

    1.5
```

Scilab ではべき乗 (x^y) の機能も備わっている。例えば $\sqrt{2}$ は

```
-->2^(1/2)
 ans  =

    1.4142136

-->2^(0.5)
 ans  =

    1.4142136
```

```
-->sqrt(2) ← 平方根を計算する Scilab 関数
 ans  =

    1.4142136
```

と計算できる。

なお，値の表示方法は特に指定していない限り 10 桁と設定されているが，format 関数を使うことで倍精度一杯，16 ケタの表示が可能となる。

```
-->2^(1/3) ← 2 の立方根を計算
 ans  =

    1.259921

-->format(20) ← 有効数字の最大桁数を 20 桁に指定

-->2^(1/3)  ← もう一度計算
 ans  =

    1.2599210498948732  ←有効数字一杯の桁数まで表示
```

$2^{1/3}$ = 1.25992104989487316476721060 72... となるので，16 桁目まで正しく，17 桁目から誤差が混入していることが分かる。

2.3 入出力

他のプログラミング言語同様，Scilab でも標準入力 (コンソールからキーボード入力) と標準出力 (コンソールへ出力) が可能である。以下，"input.sce"という名前の Scilab スクリプト (11 行分) として SciNotes から入力して適宜実行すること。なおスクリプト中のコメント文は C++ 同様，//(スラッシュの 2 連) が使用できる。

■標準入力　ユーザからの入力を待ち，入力値を変数に渡すには下記のように input 関数を用いる。**行番号を表わす"1: "は入力しないこと！**

```
1: // 入力
2: a = input("a を入力 > "); // コンソールから入力
3: b = input("b を入力 > ");
```

この結果，変数 a, b にそれぞれキーボードからの入力値が入る。

■標準出力　上記のスクリプトの 3 行目にに続いて，入力された変数 a, b}**の値を標準出力に表示するためには下記のように**\verbdisp—関数（形式指定なし），もしくは printf 関数（形式指定あり）を使用する。

```
 4: (空行)
 5: // 出力
 6: disp("a = "); disp(a);     // 改行付き文字列として出力
 7: printf("b = %25.17e\n", b); // C の printf 関数と同じ
```

上記までの部分を SciNotes で打ち込み，メニューバーから「ファイルを実行（出力なし）」を選択して実行し，a に 3 を，b に 4 を入力すると下記のような結果を得る。

```
a を入力 > 3 ← 3 をキーボードから入力
b を入力 > 4 ← 4 をキーボードから入力

 a =


    3.
b =   4.00000000000000000e+00
```

同様に，"a + b"，"a - b"の計算結果を表示させるには次のようにすればよい。

```
 8: (空行)
 9: // 計算と出力
10: disp("a + b = "); disp(a + b);
11: printf(" a - b = %25.17e\n", a - b);
```

2.4　定数と複素数の定義・演算・複素ベクトル・複素行列

Scilab に限らず，プログラミング言語等でも，よく使用する定数はあらかじめ定義され，ユーザはそれを呼び出すだけで使えることが多い。Scilab の場合，定数には % 記号を付けて他の変数と区別する。例えば円周率 $\pi = 3.141592...$ は %pi，自然対数の底 $e = 2.71828...$ は %e とする。

```
-->format(20)

-->%pi
 %pi  =

    3.14159265358979312
```

```
-->%e
 %e  =

    2.71828182845904509
```

複素数の定義に使用される虚数単位 i = $\sqrt{-1}$ も同様に定数 %i として定義されている。

```
-->%i
 %i  =

    i

-->%i^2
 ans  =

  - 1.
```

これを用いて複素数の定義もできる。

```
-->c = 3 + 2 * %i
 c  =

    3. + 2.i
```

なお，複素数定義関数 complex を用いて

```
-->c = complex(3,2)
 c  =

    3. + 2.i
```

としてもよい。

問題 2.1
$c = 2 + 3i, d = 2i$ である時，次の計算を行え。

1. ic + (3 + 2i)d
2. ($\sqrt{2}$ + $\sqrt{3}$ i)cd

2.5　ベクトルの定義

ベクトルと行列はリスト (list) というデータ型を用いて定義する。リストは大かっこ [ ] を用いて要素を囲み，カンマ (,) でデータの区切りを指定する。

```
-->a = [1, 2, 3]
 a  =

    1.    2.    3.
```

これによって横ベクトルの形式が指定できる。標準的な縦ベクトルを定義するためには，カンマの代わりにセミコロン; を用いる。

```
-->a = [1; 2; 3]
 a  =

    1.
    2.
    3.
```

ベクトルの縦横を変換する転置命令はドット＋シングルクオーテーション (.') を用いる。

```
-->a = [1; 2; 3]
 a  =

    1.
    2.
    3.

-->a.'
 ans  =

    1.    2.    3.
```

ドットなしのシングルクオーテーション (') は共役複素数化＋転置の意味になるので，複素ベクトルを扱うときには注意すること。

```
-->a = [1+%i; 2 + %i; 3+%i]
 a  =
```

```
    1. + i
    2. + i
    3. + i

-->a'
 ans  =

    1. - i      2. - i      3. - i     ←共役複素数化されて転置

-->a.'
 ans  =

    1. + i      2. + i      3. + i     ←単なる転置
```

以降，特に断らない限り，ベクトルは縦形式で設定する。
ベクトル同士の演算は実数型同様行える。例えば

$$\mathbf{v} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix},\ \mathbf{w} = \begin{bmatrix} -4 \\ -3 \\ -2 \end{bmatrix}$$

とし，$\mathbf{v}+\mathbf{w}$, $\mathbf{w}-\mathbf{v}$, $\sqrt{2}\mathbf{v}$ という計算を Scilab で実行してみよう。Scilab では $\mathbf{v}$ を vec_v，$\mathbf{w}$ を vec_w という変数に格納すると

```
vec_v = [3; 2; 1]
vec_w = [-4; -3; -2]
```

となる。演算子は実数型と同じなので

```
-->vec_v + vec_w
 ans  =

  - 1.
  - 1.
  - 1.

-->vec_w - vec_v
 ans  =

  - 7.
  - 5.
```

```
 - 3.
```

と実行すればよい。定数倍は掛け算で実行すればいいので

```
-->2^(1/2) * vec_v
 ans  =


    4.2426407
    2.8284271
    1.4142136
```

となる。

問題 2.2

1. $\mathbf{a} = [\sqrt{2}\ \ 4\ \ 5\ \ \sqrt{7}]^T$, $\mathbf{b} = [-7\sqrt{2}\ \ -4\ \ -5\ \ -6]^T$ とするとき，次の計算を Scilab で実行せよ。
   （a）$3\mathbf{a} + 2\mathbf{b}$
   （b）$(\mathbf{a} - 5\mathbf{b}) + 4\mathbf{b}$
   またこの計算を行う Scilab スクリプトファイル “prob2.2.sce”を作成し，実行せよ。
2. $\mathbf{c} = [2 + 3\mathrm{i}\ \ 2 - 3\mathrm{i}]^{\mathrm{T}}$, $\mathbf{d} = [2\mathrm{i}\ \ -3\mathrm{i}]^{\mathrm{T}} \in \mathbb{C}^2$ である時次の計算を行え。
   （a）$\mathrm{i}\mathbf{c} + (3 + 2\mathrm{i})\mathbf{d}$
   （b）$(\sqrt{2} + \sqrt{3}\,\mathrm{i})\mathbf{c}\mathbf{d}^{\mathrm{T}}$

2.6　行列の定義と演算

行列の定義もベクトル同様リストを用いる。要素はカンマで区切り，一行ごとにセミコロンを入れ，一つのリストとして定義する。例えば実正方行列 $A, B \in \mathbb{R}^{3\times 3}$ を

$$A = \begin{bmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \\ -7 & -8 & -9 \end{bmatrix}, B = \begin{bmatrix} -9 & -8 & -7 \\ -6 & -5 & -4 \\ -3 & -2 & -1 \end{bmatrix}$$

とし，これをそれぞれ mat_a, mat_b という変数に代入するには

```
mat_a = [-1, -2, -3; -4, -5, -6; -7, -8, -9];
mat_b = [-9, -8, -7; -6, -5, -4; -3, -2, -1];
```

となる。

行列の演算は，ベクトル同様の演算子が使用できる。例えば $A + B, A - B, AB$ は次のように計算する。

```
mat_a + mat_b
mat_a - mat_b
mat_a * mat_b
```

ベクトルと行列の掛け算も同様に

```
mat_a * vec_v
mat_b * vec_w
```

とすればよい。

問題 2.3

$A, B \in \mathbb{R}^{4\times4}$ が

$$A = \begin{bmatrix} 3 & 1 & 2 & 9 \\ -2 & 3 & -2 & 7 \\ 0 & -4 & 5 & -2 \\ -2 & 3 & -4 & 0 \end{bmatrix}, B = \begin{bmatrix} 4 & 2 & -7 & 3 \\ 2 & 8 & -2 & -9 \\ 0 & 4 & -2 & 2 \\ -1 & 3 & 2 & 0 \end{bmatrix}$$

であるとき，次の計算を行え。

1. $3A + 2B$
2. A^2
3. $-3A^2 + \sqrt{4}B$

2.7 ループを用いたベクトルと行列の定義

以上，Scilab におけるベクトル，行列の扱い方を見てきたが，要素をすべて決め打ちで入力してきたものばかりであった。もっと大きなサイズのベクトルや行列を扱う際には，要素の規則性を利用したり，ファイルから要素を読み込んだりする必要が出てくる。そのためには繰り返しの処理を記述するためのループを記述する。C/C++ 言語同様，do 文, while 文, for 文の 3 つのループ記述方法があるが，ここでは for 文のみを用いることにする。

for ループの使い方は

```
for 変数=初期値:(間隔値:) 終了値
  命令
end;
```

とする。間隔値を省略すると 1 ずつ増加させていくことになる。

例えば，$\mathbf{v} = [-1\ -2\ -3\ -4\ -5]^T \in \mathbb{R}^5$ というベクトルを変数 vec_v にセットするためには次のようにすればよい。

```
vec_v = [] // 空のリスト（ベクトル）
for i = 1:5
    vec_v(i) = -i;
end;
disp("vec_v = "); disp(vec_v); // vec_v を表示
```

同様に，行列も for ループを二重にすることで値の設定が可能となる。例えば $A \in \mathbb{R}^{n\times n}$ が

$$A = [-(i+j)-1]_{i,j=1}^{n} = \begin{bmatrix} -1 & -2 & \cdots & -n \\ -2 & -3 & \cdots & -(n+1) \\ \vdots & \vdots & & \vdots \\ -n & -(n+1) & \cdots & -(2n-1) \end{bmatrix}$$

という行列である時，$n = 5$ の時は次のように指定すればよい。

```
mat_a = []; // 空のリスト（行列）
n = 5; // 次元数
for i = 1:n
    for j = 1:n
        mat_a(i, j) = -(i + j - 1);
    end;
end;
disp("mat_a = "); disp(mat_a); // mat_a を表示
```

より複雑な行列 $B \in \mathbb{R}^{n\times n}$ として

$$B = \begin{bmatrix} n & n-1 & \cdots & 1 \\ n-1 & n-1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} = [n - \max(i,j) + 1]_{i,j=1}^{n}$$

を考えてみよう。定義は

```
n = 5; // 5 次元とする
b = [];
for i = 1:n
    for j = 1:n
        b(i, j) = n - max(i, j) + 1;
    end;
end;
disp("b = ");
disp(b);
```

とすればよい。

更に逆行列を inv 関数を用いて計算してみる。

```
disp("b^(-1) = ")
b_inv = inv(b);
disp(inv(b));
```

さすれば，$BB^{-1} = I_n$ となるはずである。それを確認してみよう。

```
disp("b * b^(-1) = ");
disp(b * b_inv);
```

実際にそうなっているかどうかの確認を絶対誤差を使って計算してみよう。ちなみに単位行列は eye 関数を用いて定義できる。

```
disp("|| I - B * B^(-1) ||_2 = ")
disp(norm(eye(n,n) - b * b_inv))
```

問題 2.4

ヒルベルト行列 $H \in \mathbb{R}^{n\times n}$ は

$$H = [1/(i+j-1)]_{i,j=1}^{n} = \begin{bmatrix} 1 & 1/2 & \cdots & 1/n \\ 1/2 & 1/3 & \cdots & 1/(n+1) \\ \vdots & \vdots & \vdots & \\ 1/n & 1/(n+1) & \cdots & 1/(2n-1) \end{bmatrix}$$

である。任意の $n \in \mathbb{N}$ に対して n 次のヒルベルト行列を生成するスクリプト “hilbert.sce”を作り, $n = 3, 4, 5$ として逆行列と絶対誤差も求めよ。