# 信号処理分野のための GNU Octave / MATLAB / Scilab 入門

長谷 芳樹 *

## Getting Started with GNU Octave / MATLAB / Scilab for Signal Processing

Yoshiki NAGATANI

## 0. はじめに

### 0.0. 本資料の対象者とコンセプト

• 少しはプログラミング経験があるが GNU Octave / MATLAB / Scilab は（ほぼ）初めて使う，という人を対象にしています。

• 工学的な話に特化しています。数学的な話は扱っていません。高専の電子工学科５年生を強く意識しています。

• 簡単で実用的なサンプルを集めているつもりですので，眺めるだけではなく実際に手を動かしてやってみて下さい（★は発展的内容なのでひとまず無視しても構わない）。

### 0.1. インストール（以下のいずれか一つ）

・**GNU Octave**
　`https://www.gnu.org/software/octave/` で GNU Octave 4.2.1 を入手（有用であると感じたらプロジェクトに寄附することを勧める）

・**MATLAB**
　iOS/Android 版の MATLAB Mobile でも無料で全く同じ機能が使える。特にタブレット端末ではある程度快適に利用できる。（クラウドでの実行なので端末は要オンライン。アカウント登録だけは必要だが無料なので登録しておいて損は無い。）

・**Scilab**
　`https://www.scilab.org/` で Scilab 5.5.2 を入手（6.0.0 でも構わない）（有用であると感じたらプロジェクトに寄附することを勧める）

### 0.2. 基本概念

GNU Octave / MATLAB / Scilab は以下のような特徴を持つ言語（プラットホーム）です。

• 行列演算が得意（Fortran 90 に似ている）

• 関数がとても充実しているので，車輪を再発明する前に，まずは望みの関数が用意されているかどうか探すのがよい（Octave/MATLAB では「doc［コマンド］」，Scilab では「help［コマンド］」でドキュメントを参照できる）

• 基本的に変数の「型」の概念はあまり気にしない（特に明示しなければ全ての数値は倍精度浮動小数点（double）型で扱われる）

• for 文は極力使わず，行列のままでの計算を使う方が良い（for 文は遅く，コードも長くなる）

### 0.3. 操作系

・命令の最後に「;」を付けると結果を表示しない
・カーソル↑で前のコマンドを呼び出し
・[Ctrl]+[C]で中断（Scilab では abort で復帰）

## 1. 単純な計算

### 1.1. 電卓として使う

```
>> 1+2*3    【画面に最初から表示されている >> 記号の右に半角で 1+2*3 とだけ入力して［Enter］を押す】
>> sqrt(2)
>> exp(i*pi/6)      【円周率 pi / 複素数 i（Scilab の場合は円周率 %pi / 複素数 %i）】
>> 20*log10(2)      【比 → dB の計算】
```

◆課題１：dB →比 に戻してみる（べき乗は"^"）。

◆課題２：その他，各自で好みの計算をやってみる。（利用できる演算子は 6.3.参照）

### 1.2. 変数の定義とクリア

```
>> a=1      【"=" は代入】
>> a        【変数の内容の確認】
>> b        【定義されていないとエラーになる】
>> b=2
>> a+b
>> clear a    【変数 a をクリア（メモリ解放）】
>> a        【定義されていないのでエラーになる】
>> clear    【全ての変数をクリア】
>> b        【定義されていないのでエラーになる】
```

## 2. 行列の扱い

### 2.1. 行列の基本操作

```
>> c=[10 20 30 40]【 [ ] で行列の生成や連結ができる。四角カッコは行列，と意識。列の区切りはスペース。】
>> c
>> c*2
>> c(1)   【インデックス（添え字）を指定するときは丸括弧。インデックスは 1 からはじまるので注意。従って，c(0)はエラーとなる。】
>> c(2)
>> c(5)      【定義されていないのでエラーになる】
>> c.'   【転置：1×4 の行列が 4×1 の行列になる】
>> sum(c)
>> mean(c)   【算術平均は average ではなく mean】
>> d=[50 60 70 80]
>> c+d
>> kekka=c+d    【c+d の計算結果を kekka に代入】
>> kekka      【単に c+d としたのと同じ結果になる】
>> c.*d       【要素同士の演算は"."を付ける】
>> c*d.'      【c*d（行列同士の掛け算）が可能かも試す】
>> e=[1 2 3; 4 5 6; 7 8 10]      【行の区切りは";"】
>> e^2        【行列×行列（e × e）の演算。c^2 も試してエラーとなることを確認する（Scilab 5 ではエラーにはならないがこの書き方は廃止予定）。】
```

＊ 神戸市立工業高等専門学校 電子工学科 准教授

```
» e.^2      【行列の各要素それぞれの２乗】
» inv(e)    【逆行列】
» e.'       【転置（ ' だけだと複素共役の転置)】
» sum(e)    【各列の和（行列を返す）（Scilab では sum(e,1)で列の和，sum(e,1)で行の和，sum(e)だけだと全部の要素の和（次の sum(sum(e))と同じ))】
» sum(sum(e))  【２次元行列の全部の要素の和】
» g1=[0:9]     【カッコ [ ] は無くても可】
» g2=[0:0.1:0.9]    【増分も指定できる…】
» g3=[0:9]/10       【…が，こちらの方がオススメ】
```

◆課題３：1 から 100 までの整数の合計は？

★参考：課題３（1 から 100 までの合計）は

```
goukei=0;
for n=1:100
  goukei=goukei+n;
end
goukei
```

でも計算できるが，推奨しない（遅くてコードも長い）

◆課題４：7 ～ 18 までの整数それぞれの自乗（２乗）の合計は？

### 2.2. 行列の基本関数

```
» size(c)
» size(c,1)     【行の数】
» size(c,2)     【列の数】
» ones(3,4)     【全部 1 の行列】
» zeros(3,4)    【全部 0 の行列】
» eye(3,4)      【単位行列】
```

### 2.3. 行列の一部の切り出し

```
» c(1:2)        【１次元行列の一部を切り出す】
» c_cut=c(1:2)  【切り出した部分を新しい行列に代入】
» c(2:4)
» c(2:end)      【最後の要素は end で指定可能（Scilab では $ )】
» c(end:-1:1)     【１次元行列を逆順にする】
» e(1,:)【２次元行列の一部を取り出す（「:」は全て)】
» e(:,1)
» e(2:3,1:2)     【行列の一部を取り出す】
» e(2:end,1:2)
```

### 2.4. 行列の一部への代入

```
» c(1)=15      【c の 1 つ目の要素に 15 を代入】
» c(1:2)=12    【c(1)～ c(2)に 12 を代入】
» c(1:2)=[13 14]   【c(1)に 13 を，c(2)に 14 を代入】
» c(1:2)=d(3:4)   【★代入元と代入先のサイズが同一であれば，行列の一部を行列の一部に代入することも可能。size(c(1:2))および size(d(3:4))のように代入前にサイズを確認するとミスが減る。】
```

### 2.5. 行列同士の結合・ゼロ詰め

```
» [c d]      【行列の結合（横方向に並べる)】
» [c; d]     【行列の結合（縦方向に積み上げる)】
» [c(1:3) d(4)]     【行列の一部分同士の結合】
» [c zeros(1,5)]    【ゼロの追加（右に)】
» [c'; zeros(5,1)]  【ゼロの追加（下に)】
» [c(1:2) zeros(1,5) c(3:end)]  【間にゼロを挿入】
```

★参考：最終的に作りたい行列のサイズが決まっている場合は，あらかじめゼロ行列を用意しておいてそこに代入するのが便利な場合がある（１次元行列の例）：

```
» c2=zeros(1,10)  【あらかじめゼロ行列を用意】
» c2(3:6)=c   【ゼロ行列の一部に c を代入】
» c2(3:3+size(c,2)-1)=c   【c のサイズが未知の場合にも使える，より汎用的な表現】
```

★あらかじめゼロ行列を用意しておく例（２次元行列）：

```
» e2=zeros(5,10)  【あらかじめゼロ行列を用意】
» e2(2:4,5:7)=e   【ゼロ行列の一部に e を代入】
» e2(2:2+size(e,1)-1,5:5+size(e,2)-1)=e   【e のサイズが未知の場合にも使える，より汎用的な表現】
```

★参考：2 次元配列しか入力できない関数に 3 次元配列のうちの１つのスライスだけを渡す場合などは，次元を揃える（減らす）ために squeeze 関数を用いる。

```
» sq=rand(10,20,30);
» size(sq(1,:,:)) 【３次元配列のまま】
» size(squeeze(sq(1,:,:)))【２次元配列になる】
```

★３次元配列の転置(回転)は permute でおこなえる。

```
» size(permute(sq, [2 3 1]))
```

## 3. ２次元グラフの書き方

### 3.1. グラフの基本操作

```
» x=[-50:50];【-50 から 50 まで 1 ずつ増やす】
» y=x.^2;
» plot(x,y)      【虫眼鏡アイコンで拡大できる】
» plot(x,y,'o-b')   【 o：プロット有り/ -：直線 / --：点線 / r：赤 / g：緑 / b：青 / k：黒 】
» xlabel('よこじく')   【横軸の名称】
» ylabel('たてじく')   【縦軸の名称】
```

参考：既に描いたグラフのウインドウを閉じずに新しいウインドウを作りたい場合は figure と入力すればよい（MATLAB Mobile では必須)。

### 3.2. 軸の範囲設定・グリッド表示

以下のコマンドはグラフのウィンドウが開いた状態で入力する。

```
Octave/MATLAB:
» axis([-60 60 0 3000]) 【[左 右 下 上]の順に指定】
   【xlim([-60 60]); ylim([0 3000]); でも同じ】
» grid on 【グリッドの表示】

Scilab:
» ax=get("current_axes");【軸のハンドルを取得】
» ax.data_bounds=[-60 0; 60 3000];【[左 下; 右 上]】
» ax.grid=[2 2];  【グリッドの表示】
```

### 3.3. 複数のグラフを重ねる

```
» y2=40*x-400;
» plot(x,y,'o-b')   【まずは１本目（閉じないで)】
» hold on   【既に描いたグラフを保持するように設定（hold off で解除)。Scilab の場合はデフォルトで重ね描きされる。重ね描きを解除するには set(gca(), "auto_clear", "on") とする（"off"で重ね描き)。】
```

```
» plot(x,y2,'-r')      【2本目】
» legend('1本目','2本目');   【凡例の表示】
```

### 3.4. 複数のグラフを並べる

```
» subplot(2,1,1)   【2 行 1 列に並べて 1 個目の枠に】
» plot(x,y,'o-b')   【まずは 1 本目を描く】
» title('1本目')
» subplot(2,1,2)   【2 行 1 列に並べて 2 個目の枠に】
» plot(x,y2,'-r')      【2本目を描く】
» title('2本目')
```

例：2 行 3 列の場合の通し番号（ subplot(2,3,通し番号) ):

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |

◆課題５：$-\pi \leqq x < \pi$ の範囲で $y = x$ と $y = \sin(x)$ の２つのグラフを重ねて表示してみる。（２つの関数の特徴が明確になるように $x$ の増分を充分細かく取る。また，凡例も付ける。）

◆課題６：$x > 0$ の範囲で $y = \mathrm{sqrt}(x)$ ，$y = x$ ，$y = x^2$ ，$y = \log(x)$ の４つのグラフを並べて表示してみる。（４つのグラフの特徴の違いが明確になるように $x$ の範囲と増分を調整する。また，各グラフにタイトルも付ける。）

## 4.「時刻 vs. 振幅」の信号の扱い

### 4.1. 正弦波の作成（これ以降は値はすべて SI 単位で扱う）

```
» fs=8000;  【サンプリングレート(標本化周波数) [Hz]】
» T=0.2;   【生成する信号の時間長 [s]】
» t=[0:T*fs-1]/fs;【時間軸の生成 [s] (11.4.節参照)】
» f=440;   【信号の周波数 [Hz] (440 Hz は♪ラ)】
» sig=sin(2*pi*f*t);   【正弦波関数 sin(2πft) をそのまま書いただけ ( sin の中身はラジアン [rad]) / Scilab の場合は円周率は %pi 】
```

### 4.2. 正弦波波形の表示

```
» plot(t,sig,'o-')
» xlabel('Time [s]')   【横軸の名前】
» ylabel('Amplitude [arb. unit]')   【縦軸の名前】
» title('440 Hz の正弦波')
» title(sprintf('%d Hz の正弦波',f)) 【文字列に数字を入れたい場合などはC言語の printf 形式でも文字列を生成できる（例：%05d で 5 桁にゼロ詰めした整数 / %.2f で小数部分 2 桁の小数，など)】
```

◆課題７：♪ミの音などを作成し，sig（♪ラ）と比べてみる。(sig は残しておく)

◆課題８：横軸をミリ秒にして課題７のグラフを描く（横軸の範囲も見やすく調整する：3.2.参照)。

◆課題９：同じサンプリングレート（$fs$ = 8000 Hz）で 3900 Hz の正弦波をサンプリングしたときに波形がどうなるか，グラフを描いて確認する。(sig は残しておく)

◆課題１０：sig の最大値，最小値，平均値，実効値をそれぞれ求めてみる。(いずれも一行で書ける)（ヒント：実効値(RMS) = 信号を **S**quare(自乗)して **M**ean(平均)して **R**oot(ルート)するだけ)

### 4.3. 波形の再生（MATLAB Mobile は非対応）

```
» sound(sig,fs)   【ヘッドホンの音量に注意】
» sound(sig*2,fs)   【聞き比べて違いの原因を考える】
» soundsc(sig*2,fs)   【信号が -1~1 の範囲に入るように自動でスケーリング（Scilab にはこの関数は無いので sound(sig*2 ./ max(abs(sig*2)), fs) のように自力でスケーリングする)】
```

★残念ながら本稿執筆時点では MATLAB Mobile（スマホ版）では音は出ないようであるが，MATLAB Drive に .wav ファイル等で一旦書き出して（5.1.参照）からブラウザでダウンロードして再生することは不可能ではない。（MATLAB Drive: https://drive.matlab.com/ ）

### 4.4. ノイズ(乱数)(一様分布版)

```
» r=rand(1,fs);   【1×8000 点の乱数行列】
» plot(r)
» mean(r)
» std(r)   【標準偏差。Scilab では stdev】
» median(r)   【中央値】
» histogram(r,20)   【Scilab は histplot(20,r)】
» sound(r-0.5,fs)   【直流分をカットして再生】
（※ 毎回同じ乱数列を得たい場合などには seed（種）を与える必要がある。これについては 11.5.節参照）
```

### 4.5. ノイズ(乱数)(正規分布版)

```
» r2=randn(1,fs)*0.1+0.5;【正規分布の乱数（randn 関数自体は平均 0.0，標準偏差 1.0 の乱数を返す)】
   【Scilab は r2=grand(1,fs,'nor',0.5,0.1);】
» mean(r2)
» std(r2)
» histogram(r2,20)
» sound(r2-0.5,fs)
```

★注意：一様分布の乱数を使うのか正規分布の乱数を使うのかは，用途によって使い分ける必要がある。（例えば，自然界の偶発的なノイズの模擬には正規分布乱数が適していることが多いが最大値や最小値が定義できないので困ることもある，など。）

◆課題１１：一様分布乱数から疑似正規分布風の乱数を作ってみる。

## 5. wav ファイルの読み書き

### 5.1. wav ファイルの書き出し

```
» cd 'c:¥hoge'【保存したいフォルダーを作成してから移動（この例は Windows の場合。macOS や Linux では /Users/namae/hoge/ や /home/namae/hoge/ など)】
» audiowrite('440Hz.wav',sig,fs)【Scilab では wavwrite(sig,fs,'440Hz.wav') ←順番注意】
» audiowrite('440Hz_clip.wav',sig*2,fs)
» audiowrite('white.wav',r2-0.5,fs);【0 を中心に】
```

★ステレオファイルの書き出し：

```
» audiowrite('stereo.wav',[sig' sig'*2],fs)
   【点数×2 の行列である必要がある (2×点数 では不可) ので転置する ( size([sig' sig'*2]) で各自確認を)】
【Scilab では 2×点数 の行列である必要があるので
```

wavwrite([sig; sig*2],fs,'stereo.wav') とする】

◆課題１２：各ファイルに保存された波形を Audacity / SoundEngine 等の波形編集ソフトで確認する。

### 5.2. wav ファイルの読み込み

≫ [sig2,fs2]=audioread('440Hz_clip.wav');【Scilab は wavread とする。MATLAB では.mp3, .ogg, .flac 等も読める。】
≫ size(sig2)
≫ plot([1:size(sig2,1)]/fs2,sig2)
【Scilab では行と列が逆になるので size(sig2,2) 】
≫ sound(sig2,fs2)

★ステレオファイルの読み込み：
≫ [sig_st,fs3]=audioread('stereo.wav');
≫ plot([1:size(sig_st,1)]/fs3,sig_st(:,1)) 【左 / Scilab では plot([1:size(sig_st,2)]/fs3,sig_st(1,:))】
≫ plot([1:size(sig_st,1)]/fs3,sig_st(:,2)) 【右】
≫ plot(sig_st(:,1),sig_st(:,2),'o-') 【リサージュ】

◆課題１３：任意の wav ファイルを読み込んで聞いてみる（ファイルがない場合は筆者のウェブサイトからダウンロードする（https://ultrasonics.jp/nagatani/ ））。また，サンプリングレートを変えて再生してみる。例えば，半音上げて再生するにはどうすればよいか？

## 6. 関数いろいろ

### 6.1. 最大値・最小値

≫ m=[11:20]
≫ max(m)
≫ min(m)
≫ min(17,m) 【いずれか小さい方を返す】
≫ max(13,min(17,m)) 【13 から 17 の間に収める】
≫ plot(m, max(13,min(17,m)), 'o-')

例：波形のクリップ（頭打ち）
≫ plot(t, min(1,sig*2), 'o-')
≫ plot(t, max(-0.5,min(1,sig*2)), 'o-')

◆課題１４：4.3.の「 sound(sig*2,fs) 」では何が起きていたか？ グラフを書いて確認する（できれば before と after を重ねて描いて比べる）。

### 6.2. ★検索

≫ find(m>16) 【16 より大きな配列要素のインデックス（番号）を返す（中身の値を返すわけではない）】
≫ m(find(m>16)) 【16 より大きな配列要素の値を返す】
≫ m(find(m>16))=17 【16 より大きな配列要素に 17 を代入する】
≫ find(m>16,1) 【16 より大きな配列要素の最初のインデックスを一つだけ返す（閾値を超えた判別）】

★ find 関数を使わない以下のような操作も可能である。
≫ m>16 【16 より大きい要素を 1 (true)，16 以下の要素を 0 (false) とした行列を返す（結果は元の行列と同じサイズの行列となる：find とは異なる動作）】
≫ m(m>16) 【m>16 の結果が 1(true)の要素の値を返す】
≫ m(m>16)=18 【16 より大きな配列要素に 18 を代入】

### 6.3. 演算子

算術演算子：
加算 + , 減算 - , 乗算 .* , 除算 ./ (乗算と除算にはドットが必要。ドット無しだと行列演算), べき乗 .^ , 転置 .' （ドット無しだと複素共役転置になるので注意）, 剰余 rem(a,b) （scilab では modulo(a,b)）

関係演算子：
等しい == , 等しくない ~= , より大きい > , 以上 >= , より小さい < , 以下 <=

論理演算子：
論理積(AND) & , 論理和(OR) | , 否定 ~

### 6.4. よく使う(かもしれない)関数いろいろ

zeros / ones / eye
sin / cos / tan / asin / acos / atan（全てラジアン）
log（底を e とする対数） / log10 （底が 10）/ exp
abs（絶対値） / round（四捨五入） / ceil（切り上げ）/ floor（切り捨て）/ real（実部） / imag（虚部） / sign（符号だけを取り出す）
max / min / mean / std (Scilab は stdev) / median
cd / pwd

### 6.5. ★条件分岐 (if)

条件分岐（C言語のような括弧 { } などは不要）：

```
≫ if hoge==0
    何かの処理;
  elseif hoge<10
    何かの処理;
  else
    何かの処理;
  end
```

### 6.6. ★繰り返し (for, while)

for 文：

```
≫ for n=1:10
    何かの処理;
    if ループ強制終了の条件
      break
    end
    if ループ反復の条件
      continue
    end
  end
```

while 文：

```
≫ n=1;
≫ while n<=10
    何かの処理;
    n=n+1;
  end
```

★ for 文と if 文の例（100 より大きな 13 の倍数を小さい順に 5 個見つけるプログラム）：

```
mitsuketa = 0;          % カウンターの初期化
for n=101:200
  if rem(n,13) == 0     % 剰余がゼロの時
    disp(n)             % 結果の表示
    mitsuketa = mitsuketa + 1;
  end
  if mitsuketa >= 5
    disp('5 個見つけたので終了します')
    break               % for ループを抜ける
  end
end
```

ちなみに上記のプログラムと同じことが，

```
n=[101:200];
n(find(rem(n,13)==0,5))
```
の 2 行だけで実現できる（find 関数で n の行列から 13 で割った余りが 0 のものを見つけ出して最初の 5 個分のインデックスを返す。その結果を n( )の添え字に入れることで n の要素の値が表示される)。このように，Octave/MATLAB/Scilab では for 文や while 文の使用は可能な限り避けるのが望ましい。

### 6.7. ★差分・累積和

```
» s=[0 1 1 2 3 5 8 13]
» diff(s)        【差分（要素数は 1 つ減る)】
» cumsum(s)      【最初の要素からの累積和】
```

★参考：行列表現で差分を実現する方法

```
s(2:8)-s(1:7)【隣り合う要素の差 (diff(s)と同じ)】
s(2:end)-s(1:end-1)  【より汎用的な書き方】
```

### 6.8. 行列を２次元画像として表示

```
» image([1:64])  【Scilab は Matplot([1:32])】
» image(eye(100,150)*16+randn(100,150)*8+32)
```
【★ imagesc 関数を使うとカラーマップ（グラデーション）の全範囲の色を使用するように自動的にスケーリングされる。Scilab の Matplot 関数にはこの機能は無いので自力でスケーリングする必要がある。】

## 7. 信号処理

### 7.1. FFT

```
» u=sin([0:0.5:10]);  【なにか適当な信号を用意】
» fft(u)              【FFT 自体はこれだけで OK】
» plot(abs(fft(u)),'o-')  【FFT 結果の振幅スペクトル表示（FFT の結果をそのまま表示するだけでは，横軸は Hz 等ではなく「点数」となる → 課題１５参照)】
```

参考：Octave/MATLAB/Scilab の fft 関数は任意の点数の信号に適用可能（$2^n$ 点である必要は無い）

例：ひずんだ正弦波の振幅スペクトルとの比較

```
» subplot(2,1,1)
» plot(t,sig,'o-b')        【4.1.で作った sig】
» sig_clip=min(1,max(-0.8,min(0.8,sig)));
» hold on
» plot(t,sig_clip,'o-r')  【クリップ波形】
» legend('正弦波','クリップされた正弦波');
» title('波形')
» xlabel('時刻 [s]'); ylabel('振幅[arb. unit]');
» subplot(2,1,2)
» plot(abs(fft(sig)),'o-b')  【元の波形】
» hold on
» plot(abs(fft(sig_clip)),'o-r')  【クリップ波形】
» legend('正弦波','クリップされた正弦波');
» title('振幅スペクトル')
» xlabel('点数 [点]'); ylabel('振幅スペクトル');
```

◆課題１５：4.1.で作成した sig（あるいは上記の例で作成した sig_clip）の FFT 演算をおこない，振幅スペクトルを正しい横軸（点数ではなく Hz）で表示する。【ヒント：FFT の周波数解像度は $1/T$ [Hz]（$T$ は信号の全長 [s]），FFT 結果の点数は FFT 前の時間波形の点数 $N$ と同じ】

◆★課題１６：440 Hz の正弦波を
（A）$fs$ = 8 kHz，$T$ = 0.01 s
（B）$fs$ = 10 kHz, $T$ = 0.008 s
の各条件で作成し，１つのグラフに表示する。縦軸の dB 表記（最大値で正規化）も試す。また，このとき，それぞれのピーク値の周波数はどうなるか？

◆★課題１７：一様分布のノイズあるいは正規分布のノイズ（4.4.および 4.5.参照）のスペクトルを調べてみる。また，インパルスのスペクトルも調べてみる。

### 7.2. ★たたみ込み

```
» c1=[0 0 2 0 0 0 0 0 0 3 0 0 0 0 -2 0 0 0 0 0];
» c2=exp(-[0:9]);
» c3=conv(c1, c2);
» subplot(3,1,1); stem(c1,'o-'); title('入力');
» subplot(3,1,2); plot(c2,'o-'); title('IR');
» subplot(3,1,3); plot(c3,'o-'); title('出力');
```
（Scilab には stem は用意されていないので plot で）

◆課題１８：任意の音声や音楽に残響をたたみ込んで聞いてみる。（ファイルがない場合は筆者のウェブサイトからダウンロードする）

★例：ノイズだけでドラム演奏の模擬音を作ってみる

```
» fs=8000; 【サンプリングレート】
» gakufu = zeros(1,fs);  【1 秒分のドラム演奏楽譜】
» gakufu(round(0.2*fs))=1.0; 【時刻 0.2 秒の時点で強さ 1.0 でドラムを叩く】
» gakufu(round(0.4*fs))=0.5; 【時刻 0.4 秒】
» gakufu(round(0.5*fs))=0.5; 【時刻 0.5 秒】
» gakufu(round(0.6*fs))=2.0; 【時刻 0.6 秒】
» subplot(4,1,1); plot(gakufu); title('楽譜');
» SN=exp(-[0:fs-1]/500).*(rand(1,fs)-0.5);【ドラムのスネアっぽい音（ホワイトノイズが指数関数で減衰する)】
» subplot(4,1,2); plot(SN); title('スネア単音');
» sound(SN, fs);
» deoto=conv(gakufu, SN);【畳み込み】
» subplot(4,1,3); plot(deoto); title('出音');
» sound(deoto, fs);
```

### 7.3. ★相互相関

ドラム演奏の模擬音とスネア単体の音の相互相関を計算することで楽譜情報を抽出（推定）する：

```
» gakufu2=xcorr(deoto, SN);  【相互相関】
» subplot(4,1,4);plot(gakufu2);title('推定結果');
```

◆★課題１９：推定した楽譜情報にノイズが含まれる理由考える。【ヒント：作成したスネアの音の特徴は？】

## 8. 自作スクリプト・自作関数

### 8.1. スクリプトの作成と実行

処理を上から順次実行するだけの場合（引数や返り値やローカル変数の使用が必要ない場合）は，単にテキストファイルにコマンドを列挙するだけで良い。

- edit を実行すればエディタが開く。
- Octave/MATLAB の場合は拡張子.m，Scilab では拡張子.sci で保存する。いずれもファイル名は半角英数のみ。また，ファイル名の１文字目に数字は使えない。
- 実行方法：任意のフォルダーに保存（半角英数のみ）

して [F5] を押す。あるいは，コマンドで実行する場合やスクリプトからスクリプトを呼び出す場合は，そのファイルの存在するフォルダーに cd コマンドで移動するか Path を通してから以下を実行する：

Octave/MATLAB：単にファイル名（.m を除いた前半部分のみ）を入力する

Scilab：exec('ファイル名.sci') と入力する。（もしくはフルパスで exec('c:¥hoge¥ファイル名.sci') と指定すればフォルダー移動や Path 設定は不要である）

### 8.2. 関数の作成と実行

返り値が必要な場合やローカル変数を用いる場合（関数化）は以下のようにおこなう。（★関数の中で定義した変数はローカル変数となる。変数のスコープについては MATLAB/Scilab のドキュメントの global の項目参照。）

#### Octave/MATLAB の場合：seigenha.m

```
function [t_axis,waveform] = seigenha(f,T,fs)
% コメント行は % です
      t_axis = [0:T*fs-1]/fs;
      waveform = sin(2*pi*f*t_axis);
```

参考：Octave/MATLAB では ファイル名.m と関数名が一致する必要がある。作ったファイルの存在するフォルダーに移動するか Path を通してから実行する（ファイル名の.m を除いた前半部分＝関数名 を入力する）。

実行方法：

```
>> [t,sig] = seigenha(440, 1, 8000);
>> sound(sig, 8000);
```

#### Scilab の場合：seigenha.sce

```
function [t_axis,waveform] = seigenha(f,T,fs)
// コメント行は // です
      t_axis = [0:T*fs-1]/fs;
      waveform = sin(2*%pi*f*t_axis);
endfunction
```

参考：Scilab では ファイル名.sce と関数名は一致していなくてもよい。

実行方法：

```
>> exec('c:¥hoge¥seigenha.sce');【関数の読み込み（ファイルを書き換える度に読み込みなおす）】
>> [t,sig] = seigenha(440, 1, 8000);
>> sound(sig, 8000);
```

◆課題２０：周波数 $f$c［Hz］の搬送波（正弦波）を周波数 $f$s［Hz］の信号で変調度 $m$ で振幅変調する関数を作成し，波形と振幅スペクトルを確認する。

## 9. ファイルの読み書き

### 9.1. CSV 形式(テキストファイル)の読み書き

カンマ区切りファイル（列をカンマで区切ったファイル）を作り，保存したフォルダーに移動してから下記を実行する（もしくはファイル名を 'c:¥hoge¥data.csv' のようにフルパス指定してもよい）：

```
>> X=csvread('data.csv');  【Scilab は csvRead】
>> plot(X(:,1),X(:,2),'o-')
>> csvwrite('data2.csv',X)  【Scilab は csvWrite(X,'data2.csv')】
```

参考：任意の区切り文字（カンマ以外）で区切られたファイルを読み書きする場合は dlmread/dlmwrite が便利（Octave/MATLAB の場合）。

```
>> X=dlmread('data.txt',' ');  【スペース区切りの場合 / Scilab では csvRead('data.txt',' ')】
```

### 9.2. ★ MATLAB 形式/Scilab 形式の読み書き

```
>> save('data1.mat')  【全変数を保存 / Scilab では savematfile で MATLAB 互換ファイルが作成できる。save('data1.dat')では Scilab 独自のバイナリ形式で保存される。】
>> save('data2.mat','a')  【変数 a を保存】
>> save('data3.txt','a','-ascii')【テキストで保存】
>> load('data1.mat')【全変数を読み込み / Scilab で MATLAB 互換ファイルを読み込む場合は loadmatfile】
>> load('data2.mat','a')  【変数 a のみを読み込み】
```

## 10. 総合課題

◆課題２１：信号 u（7.1.節）にハン窓（ハニング窓）などを掛けて FFT し，矩形窓（窓なし）と結果を比較する。【ヒント：Octave/MATLAB では hann 関数も利用できますが，ハン窓の数式 0.5*(1-cos(2πn/N)) をそのまま使ってみるのもよい練習になるでしょう（いずれにせよ１行で書ける）】

◆課題２２：信号 u（7.1.節）をそのまま FFT した結果と，信号の後ろにゼロ詰めしてから FFT した結果とを比較する。【ヒント：波形の後ろに任意の点数（通常は元信号の整数倍を選ぶことが多い）のゼロ行列を追加すると，信号の全長 $T$［s］が変わるので，それに応じて周波数解像度 1/$T$［Hz］も変化します（課題１５も参照）。】

◆課題２３：スイープ音（連続的に正弦波の周波数が変化する）を作成して聞いてみる。また，そのスペクトルも確認する。【ヒント：周波数が一定の正弦波ならば位相［rad］は時間に正比例しますが，周波数が時間的に変化する場合には位相の増え方がどうなるかを考えてください。いずれにせよ単調増加であることには違いありませんが，正比例（一次関数）ではないはず。ついでに指摘しておくならば，sin( ) 関数の中身は位相［rad］ですね。】

◆課題２４：課題２３で作成した信号のスペクトログラムを表示する。【ヒント：スペクトログラムというのは短時間フーリエ変換（解析範囲を区切った FFT）を単純に時間方向に並べたものですから，ひたすら横移動（時間窓の移動）を繰り返しながら FFT を実行し続けるだけです。例えば結果を 6.8.で用いた image 関数で表示できる形式で格納しておけば手軽に可視化できるでしょう。】

◆課題２５：JIS TI201-1:2000（オージオメータ）準拠の断続音を作成する（テーパーは０～１までのリニア変化でＯＫ）。【ヒント：瞬時にいきなり立ち上がる正弦波は「プチ」という音が聞こえてしまいます（単に数学的に高周波成分が含まれるというだけでなく，実際に耳で聞

こえますので試してみて下さい)。この「プチ」音を避けるためには，滑らかに立ち上がって滑らかに立ち下がるような処理が必要です。これを音響学業界では「テーパー」などと呼びます（業界によって名称は異なる)。】

◆課題２６：インパルス，もしくは 4.5.で作成したホワイトノイズに LPF（低域通過フィルター)，HPF（高域通過フィルター)，BPF（帯域通過フィルター）などを適用して，波形の変化を観察するとともに，聞き比べる。また，その振幅スペクトルを表示して確認する。さらに，気に入った音ができたら 7.2.のようにドラム演奏のように仕立て上げるのも面白い。【ヒント：フィルターについてはここでは深入りはやめて，ひとまず，通過させる周波数以外はバッサリと除去する（ゼロにする）フィルターを作成してみて下さい。その場合にフィルタリング後の波形がどのような特徴を持つかについても検討してみると理解が深まるはずです。】

# 11. もう少し発展的な技術

## 11.1. ★ウィンドウの位置とサイズの指定

Octave/MATLAB の場合：

```
>> fig1=figure(1)  【新しいウィンドウを開く】
>> set(fig1,'Name','ウィンドウのタイトル（任意の文字列）','position',[10 100 800 600])  【横 800px ×縦 600px のサイズに設定。ディスプレイの左下の隅を(0,0)として，ウインドウの左下の座標を(10,100)に移動】
```

Scilab の場合：

```
>> fig1=figure('Figure_name', 'ウィンドウのタイトル（任意の文字列）', 'position', [0 0 800 600], 'BackgroundColor', [1 1 1]);【横 800px ×縦 600px のサイズに設定。ディスプレイの左上の隅を(0,0)として，ウインドウの左上の座標を(0,0)に移動】
```

## 11.2. ウィンドウ(グラフ)を画像として保存

Octave/MATLAB の場合：

```
>> saveas(figure(1),'hoge.png','png');  【ウインドウを開いたときの「figure(1)」の数字と揃える】
```

Scilab の場合：

```
>>xs2png(gcf(),'hoge.png');  【対象のウィンドウを scf(1)で開いた場合は xs2png(1,'hoge.png')のように数字を揃える】
```

## 11.3. ★グラフの描画タイミングの指定

• 重い処理を繰り返し実行している場合や何十本の波形を一気に描いたりする場合に，処理が終わるまで画面が更新されない場合や，逆に画面の更新に時間が掛かって処理が遅くなる場合がある。その場合は画面の更新タイミングを手動で指定することで対処できる。

Octave/MATLAB の場合：

```
>> drawnow      【これで強制的に描画する】
   （Octave/MATLAB では基本的に処理が終わるまでは描画されない）
```

Scilab の場合：

```
>> drawlater()  【drawnow() を実行するまで描画を停止】
>> drawnow()    【これで強制的に描画する】
```

## 11.4. ★時間軸の生成時の流儀

時間軸の生成方法（4.1.節参照）には諸派が存在するが，どれかひとつだけが正しいというわけではない。各自，好みのものを見つけて欲しい（本資料では A のパターンを採用している)。ただし，いずれの場合も，要素数（点数）が何点になるのかを常に意識していないとトラブルのもととなる。

A) `t=[0:T*fs-1]/fs;`  【`t=[0:floor(T*fs)-1]/fs;`と等価（`floor` は小数部分の切り捨て)】
B) `t=[1:T*fs]/fs;`  【A と似ているが `0` から始めないバージョン】
C) `t=[0:round(T*fs)-1]/fs;`  【`T*fs` の小数部分が `0.5` より大きい場合は A よりも要素数が 1 つ増える。`T*fs` がちょうど整数になる場合は A と全く同じ。】
D) `t=[1:round(T*fs)]/fs;`  【C の `0` から始めないバージョン】
E) `t=[0:1/fs:T];`  【増分を `1/fs` としているので直感的に理解できるが，要素数が 1 点増えるので注意。】
F) `t=[0:1/fs:T-1/fs];`  【E を A と結果が同じになるように調整したもの】

## 11.5. ★乱数生成時の注意

4.4.節および 4.5.節で紹介した rand 関数および randn 関数は実行するたびに異なる結果を出力する。通常はこれで問題がないが，実験の再現性（同じ実験をやり直したときに同じ結果が得られるか）を確認したい場合などには逆にこれが問題となることがある。この場合は乱数の seed（種）を与えることで再現性が得られる。

なお，Scilab の場合は逆に，起動するたびに同じ系列にリセットされる（起動するたびに毎回同じ乱数が出力される）ので，これを避けるためには rand 関数を実行する前に rand('seed', getdate('s')) のようにランダムな seed を与える必要がある（この例では getdate('s') で得られる時刻情報（秒）を事実上ランダムと見なせる値として用いている)。

```
>> rand('seed',任意の整数)【MATLAB の場合は rnd(任意の整数) でもよい。ただし，同じ seed 値を与えた場合でも Octave / MATLAB / Scilab の間では同じ結果は得られない。】
```

例：

```
>> rand('seed',123)【任意の整数として 123 を選んだ】
>> rand(1,3)        【３つの乱数(※)を得る】
>> rand(1,3)        【異なる３つの乱数が得られる】
>> rand('seed',123)  【再び 123 を与える】
>> rand(1,3)        【※と同じ３つの乱数が得られる】
```

## 11.6. ★変数名を変数で指定したい場合

変数名が通し番号になっているような場合には eval 関数が便利な場合がある。eval 関数では，文字列が式として実行される。ただし，eval 関数を多用すると可読性やデバッグのしやすさが著しく低下するので使用は極力避けることを勧める。

```
>> hensuu001=123;
>> hensuu002=456;
>> hensuu003=789;
>> for num=1:3
     eval(sprintf('hensuu%03d',num))
   end
```

## 12. さらに発展させたい方へ

### 12.1. 情報の調べ方

Octave/MATLAB/Scilab はドキュメントが充実しているため，かなりの作業を容易に進めることができる。doc コマンド（Scilab では help コマンド）で呼び出せるドキュメントに加えて，ウェブでも参照することができる。さらに，各関数のページには，文法や使い方の実例に加えて，関連する関数などがページの最下部に列挙されているので大いに参考になる。

```
MATLAB:
  https://jp.mathworks.com/help/
（Octave の基本的な関数は MATLAB 互換であるので，多くの場合で MATLAB のドキュメントが参考になる。）
Scilab:
  https://help.scilab.org/docs/5.5.2/ja_JP/
（上記はいずれも日本語化されているが，一部の新機能等はまだ翻訳されていない場合もある）
```

### 12.2. 信号処理のサンプルプログラム

Octave/MATLAB/Scilab で動作する信号処理のデモプログラムを下記で公開している。いずれも実用的な信号処理手法を視覚的に理解できるように紹介しているので参考にして欲しい。なお，Octave/MATLAB 版は日本語版も用意している。

```
https://github.com/nagataniyoshiki/SignalProcessingDemo
```

### 12.3. Raspberry Pi への Octave のインストール

通常の PC 上の Linux のみならず，Raspberry Pi（Raspbian）上でも学習用途には充分な速度で Octave が動作するので，極めて安価にプログラミング環境が構築できる（ただし残念ながら本稿執筆時点のバージョンでは sound 関数が実装されておらず，音は出ないようである）。

```
Raspbian（Raspberry Pi）/ Ubuntu / Debian 等:
  sudo apt-get install octave
CentOS / Fedora 等:
  sudo yum install octave
```

### 12.4. MATLAB の購入

以前から Scilab は高いクオリティーを実現しているし，GNU Octave も近年急速に安定感を増しているのは間違いない。しかしながら，やはり商用ソフトウェアである MATLAB にはその価格に見合った価値があるのもまた否めない事実である。さらに，有効なライセンスを所有している場合にはブラウザから実行できる MATLAB Online が利用できる（無償の MATLAB Mobile とは別物）ので，これも有用である。

まずは所属する教育機関が MATLAB TAH Student ライセンスを契約しているかどうかを確認する。契約している場合には無償で利用できる。ただしこのライセンスは卒業（退職）後はアンインストールしなければならない。

MATLAB を購入したい場合は，学生の場合は Student 版が極めて安価（¥4,990）で購入できる（Academic 版ではなく Student 版）。なお，MATLAB を MATLAB らしく使うためには最低でも Signal Processing Toolbox だけは必須だが，これも Student 版には標準で付いてくる。とはいえ各種 Toolbox も同時購入の場合はたったの ¥999 なので，各自が使いそうなものを任意に選んでおけばよい。ただし，このライセンスをラボの研究費などで購入することは禁止されているので，自費で購入していただくしかない。また，学生の私物の PC にしかインストールできない。

```
Student 版（MATLAB Student）の購入先：
  https://jp.mathworks.com/academia/student_version.html
```

残念ながらあなたが学生ではない場合でも，Home ライセンスが ¥16,500 で購入できる（事実上必須の Signal Processing Toolbox が ¥4,990 なので合計 ¥21,490）。ただし，このライセンスは研究や業務には使用できない。

```
Home ライセンスの購入先：
  https://jp.mathworks.com/pricing-licensing.html?intendeduse=home&prodcode=ML
```

余談だが，研究室などで研究用途で使うための通常のライセンスは，アカデミック価格であっても MATLAB 本体が 7 万円程度，Signal Processing Toolbox が 3 万円程度なので，合計 10 万円を超える。自宅の PC で使える学生版の購入を勧める理由がわかって頂けることと思う。（価格は全て本稿執筆時点（2018 年 1 月））


## 謝辞

本資料はチューリッヒ大学の橘亮輔氏が執筆されたテキスト「MATLAB 基礎」に触発されて作成したものである。ここに深く敬意を表す。また，神戸市立工業高等専門学校電子工学科の尾山匡浩氏，阿南愛氏，谷川楓夏氏，神戸市立工業高等専門学校専攻科電気電子工学専攻の吉野寿紀氏，後上正樹氏，電気通信大学情報理工学研究科の饗庭絵里子氏，豊橋技術科学大学大学院工学研究科の松井淑恵氏，東京工業大学の小野寺輝子氏，および，秋山大知氏には原稿について貴重なコメントを頂いた。ここに謝意を表する。


## 本資料について

本資料の電子ファイルおよび課題の解答例を筆者のウェブサイトで公開予定である。個人や研究室などでぜひご活用いただきたい。

```
筆者のウェブサイト：
  https://ultrasonics.jp/nagatani/
```

加えて，本稿についてのアイデアやご意見等をお持ちの場合や再配布や改変などを希望される場合は筆者までご連絡いただきたい。できる限り柔軟に対応する。