MATLAB 演習（１）(13/11/13)

今回から数学ソフト MATLAB を用いた音響データの処理に関する演習を行う。これには音を作る処理（音響合成）と音を調べる処理（音響解析）の２つの面があるが、当面は音響合成を扱う。以下では MATLAB で音響データを扱う基本事項について記す。

1 MATLAB について

MATLAB は数学の数値計算、グラフや図形の描画、数式処理機能（部分的）を備えた数学ソフトで、信号処理関係のアプリケーションに重用されている。

基本的な使い方については以下の URL にある資料類を参照。

`http://www.slis.tsukuba.ac.jp/~hiraga/analysis2/resource/tools.shtml`

ただしこれらの説明書は違う計算機システム上での使用法を記したものであり、それを現システム用に更新していないので、そういった部分は修正する必要がある。全学システム上の `hiraga.yuzuru.gf/public` の下、`doc` ディレクトリには関連文書類を、`matlab` ディレクトリの下には MATLAB プログラム例（M-file）をおいておく。

なお数学計算や信号処理に使われるソフトとしては他に Mathematica、Scilab（河辺先生の実験で使用）などがあるが、ここでは触れない。

2 MATLAB プログラミング

多くの人にとっては MATLAB は事実上、初めて使うプログラミング言語だろうから、まずは使い方やプログラミング方法に慣れることが先決である。詳しい言語仕様は別途講義形式でも述べていくが、最初のうちはとりあえずプログラムの実例を打ち込んで動かしてみる、それをいろいろ変更・追加していって自分でプログラムを書けるようにしていくことを心がける。

上記のページに掲げた資料類のうち、とりわけ「MATLAB 簡易マニュアル（中野倫靖）」にはよく目を通し、またそこに書いてある例を実際に実行してみること。

MATLAB のプログラムは、見かけでは C のプログラムと似ている点が多い（３編生も含めて、C プログラミングの知識があることは前提とする）。しかし一方、いろいろな点で違いもあり、C のつもりで書いていたらエラーになるケースも多いので注意が必要である。

2.1 ヘルプ、表示機能の制御

```
help 項目名
helpwin
helpdesk
demos
```

なにはともあれ、わからないことがあったら「`help 項目名`」を実行してみること。項目名には関数名などを書く。ただし、項目名がわからない場合も多い。その場合には「`helpwin`」とすれば別ウィンドウが開き、内容検索などもできる。「`helpdesk`」は入門用ドキュメントにアクセスできるので、まずはここから「MATLAB → Getting Started」あたりにひと通り目を通すこと。「`demos`」は各種デモプログラムへのアクセスページである。`helpwin, helpdesk, demos` は同一ウィンドウに開かれる。相互に行き来することも可能である。

`help` の表示、あるいは計算の実行結果などは、デフォールトではウィンドウにそのまま流され、量が多いとウィンドウ上部から消えてしまう。右側のスクロールバーで再表示させることもできるが、画面が一杯になったらいったん表示を停止し、続きを後から表示させることもできる。そのためには「`more on`」とすれば、以後の表示は画

面が一杯になったところでいったん停止し、左下に「-- more --」と表示される。ここで「return (enter, 改行)」キーを打てば１行分、空白キーを打てば１画面分表示が進む。“q” と打てば表示（を実行しているコマンド）を強制終了する（UNIX の more, less コマンドと同じ機能）。more の機能を取りやめるには「 more off」とすればよい。詳しくは「help more」参照。

more に限らず、MATLAB の多くの制御コマンドは on/off によって機能を制御できる。

2.2 コマンド履歴

コマンドウィンドウで打ち込んだコマンド列は「コマンド履歴」に保存される。コマンド履歴を表示したり非表示にするには、メニューの「デスクトップ」から「コマンド履歴」を選択する。

実習中に打ち込んだ実行列はコマンド履歴に保存されているので、復習などをするためにはそれを参照すればよい。またコマンド履歴の内容をファイル（M-ファイル）に保存することもできる。

2.3 計算の実行と表示

C がコンパイラ言語であるのに対し、MATLAB はインタプリタ言語で、C と違って打ち込んだコマンド（C の実行文に相当）はその場で計算される（C ではプログラムをいったんコンパイルしなければならない）。

コマンドの末尾に “;”（セミコロン）を 入れなければ 計算結果が表示される。一方、セミコロンを入れればそのコマンドの計算結果は表示されない（C では実行文の末尾には必ずセミコロンを入れなければならない）。したがって、インタラクティブに計算を行っているときはセミコロンなしで結果を表示させればよい。一方、計算結果が大きな配列（ベクトル）だったりすると、表示量が膨大になるので、セミコロンをつけて表示を行わず、必要な部分だけを取り出して表示させればよい。

特にプログラム中では、結果をいちいち表示させるとうるさいので、原則として各コマンド末尾には必ずセミコロンをつける。反面、プログラムにエラーがある場合などにはセミコロンなしの行を入れておけば、その時点での値を表示でき、デバッグに利用できる。

結果の表示は、変数への代入文の場合：

```
>> a = sum(1:10)
   a = 55
```

（1 から 10 までの和：次項 2.6 も参照）のように、結果が変数への代入文の形で表示される（実際には前後にいくつか改行が入るが、間延びするので上では詰めて書いてある）。一方単なる計算式の場合：

```
>> sum(1:10)
   ans = 55
```

のように、ans という変数への代入文の形で表示される。ans は特別な変数で、（代入文でない）直前の計算結果が保持される。したがって計算を実行するごとに値が変わる。結果を保存しておくには、「a = ans」のように、他の変数に値を代入すればよい。

2.4 数値データ

MATLAB では C と同様、数値データの表現として整数型と（倍精度）実数型とがある。ただし、変換は自動的に行われるので、C のように変数の型を区別する必要はない。必要なら、整数型⇔実数型の相互変換もできる。

数値（特に実数値）の表示形式は format コマンドで指定できる。デフォールトでは format short で小数点以下４桁が表示されるが、format long とすれば１５桁（実際の計算精度）の表示ができる。詳しくは「help format」参照。

2.5 文字データ、文字列

C では文字型データは 'c' のように single quote で囲って表し、文字列（string）は "string" のように double quote で囲って表す。これに対し、MATLAB ではどちらも 'c', 'string' のように single quote で囲って表し、文字型データと文字列とは区別しない。

2.6 ベクトル（と行列）

MATLAB の基本となるデータ型はベクトル（及び行列）で、ベクトルは C の 1 次元配列、行列は 2 次元配列にあたる。ただし：

- ベクトルは行列の特別な場合なので、横ベクトル／縦ベクトルの違いがある（下記参照）。
- ベクトル・行列の要素（＝配列要素）は C のような a[5], b[2][3] のような形ではなく、丸カッコを使って a(5), b(2, 3) のように参照する。
- ベクトル要素の添字は 1 から始まる（C では配列の最初の要素の添字は 0）。
  つまり C では a[0] が最初の要素、MATLAB では a(1) が最初の要素である。
- C と違って、MATLAB ではベクトル・行列に直接演算を適用できる。

本実験の範囲では、主としてベクトル（横ベクトル）を使用し、行列はあまり使わない（ただし、例えばステレオ音を作るには、$2 \times n$ 行列が必要となる）。そこで横ベクトルの作り方や扱いを簡単に見ておく。

- 直接作る
  横ベクトル（$1 \times n$ 行列）は、要素を角カッコ [...] で囲って表わせる。

  ```
  >> a = [1 2 3] % [1, 2, 3] のように要素をカンマで区切ってもよい。
  ```

  （注： MATLAB では % があると、その行の残り部分はコメントと見なされる。）
- a = zeros(1, n)
  要素がすべて 0 の、長さ n のベクトルを作る。n は実際の数値、あるいはあらかじめ値を代入しておいた変数、さらには計算式でもよい。
- a = ones(1, n)
  要素がすべて 1 の、長さ n のベクトルを作る。
- a = 1:n
  差分列 [1 2 3 ... n] を作る（n が 1 以上の整数）。
  一般には n:m は n から m までの差分列になる。
  n が小数点以下のある実数の場合には、小数点以下を切り捨てた整数のところまでの列が作られる。また n が 1 より小さい場合には空列 [ ] となる。
- c = a:d:b
  d を増分とする差分列を作る。例えば 0:0.1:1　なら、[0 0.1 0.2 ... 0.9 1] という、長さ 11 のベクトルができる。
  $a < b$ なら $d > 0$、$a > b$ なら $d < 0$ でなければならない。
- 連結
  2 つのベクトル a, b をつないだベクトルを作るには、[...] で囲って [a b] とすればよい。例えば [[1 2 3] [4 5 6]] は [1 2 3 4 5 6] になる。

長さが同じベクトル同士は直接演算ができる。例えば a = [1 2 3]; b = [4 5 6] なら、

```
>> a+b
   ans = [5 7 9]
```

となる（実際の表示では両側の [ ] は表示されない）。差についても同様。

ただし、要素同士の積・商・べき乗については、演算子は *, /, ^ ではなく、.*, ./, .^ のように、前にピリオドをつけなければならない。これは、*, /, ^ は行列の積・商（逆行列との積）、べき乗と扱われるからである。

```
>> a .* b
   ans = [4 10 18]
>> a * b
   (エラーになる)
```

例：「0:0.1:1」は「(0:10)./10」としても同じ（定数との積・商は「(0:10)/10」のようにピリオドなしでもよい)。

練習： 次のそれぞれはどのようなベクトルになるか。

- (0:0.1:1).^2
- 1./(1:10)

MATLAB にはベクトルを操作するための様々な演算子や機能が用意されており、うまく活用すれば（C のような手続き型言語で書く場合に比べて）コンパクトでわかりやすいプログラムが書ける。

- 多くの関数は単一の数値だけでなく、ベクトルを引数として受け取り、各要素の関数値のベクトルを値として返す。例えば：

  ```
  >> t = 0:pi/180:2*pi;
  >> y = sin(t);
  ```

  とすれば、1 周期（2*pi、つまり 2π）を 360 等分（つまり 1 度単位）したデータ列 t に対し、サイン関数 sin(t) の値をベクトルとする関数を返す。そこで：

  ```
  >> plot(t, y);
  ```

  とすれば、$y = \sin t \ \ (0 \leq t \leq 2\pi)$ のグラフを描くことができる。（グラフについては 2.10 も参照）

  注： ベクトル t の要素数は 361 であることに注意。

- 例えば $1 \sim N$ の和を求める処理は、C 流に書けば次のようになる。

  ```
  >> N = 100; s = 0;
  >> for n=1:N;
  >>      s = s + n;
  >> end;
  ```

  これに対し、ベクトル演算と sum 関数を使えば：

  ```
  >> N = 100;
  >> s = sum(1:N);
  ```

  のように簡単に書ける。

2.7 応用例：音データの連結と保存

例えば ssyn 関数（5 参照）を使ってドレミの 3 音（CDE の 3 音）は、次の MATLAB コマンド列で作成できる（Fs（サンプリングレート）は ssyn 標準値の 44,100 Hz、長さは各 1 秒とする。なお数値 60, 62, 64 は MIDI ノート番号である。

```
>> Fs = 44100;
>> y1 = ssyn(440*2^((60-69)/12));
>> y2 = ssyn(440*2^((62-69)/12));
>> y3 = ssyn(440*2^((64-69)/12));
```

これを順に演奏するには、個別に wavplay を呼び出して：

```
>> wavplay(y1, Fs);
>> wavplay(y2, Fs);
>> wavplay(y3, Fs);
```

とするのでもよいが、次のようにデータを１つにつなげば「１曲分のデータ」になる。

```
>> y = [y1 y2 y3];
>> wavplay(y, Fs);
```

あるいは直接

```
>> wavplay([y1 y2 y3], Fs);
```

参考： うるさいことを言えば、各音の先頭と末尾のデータは時間的に重複している可能性がある。その場合は下のように重複分を取り除いた方がいい。

```
>> y = [y1(1:(length(y1)-1)) y2(1:(length(y2)-1)) y3];
```

また１音ごとに曲データにつないでいくには：

```
>> y = [];
>> for n=1:10;
>>     y = [y ssyn(...)];
>> end;
```

のように、繰り返し文の中で y をつないでいけばよい（ただしこの方法はメモリ使用にムダがあるため推奨されない）。

音データをつなぐと、音同士の継ぎ目のところで「プチ」という雑音が入る可能性が高い。これは前後でデータが不連続になるためである。これを避ける方法はいろいろあり、例えば音の後ろのほうの音量を絞ってしまうのはその１つである（これについては後で述べる）。

「曲データ」ができれば、それを（WAV 形式で）ファイルに保存すれば、MediaPlayer などのオーディオ再生ソフトで聴くことができる。WAV ファイルへの保存には wavwrite 関数を用いる。例えば音データ y, サンプリングレート Fs の曲を “kaeru.wav” というファイルに保存するには、下のようにすればよい。

```
>> wavwrite(y, Fs, 'kaeru.wav');
```

2.8 関数ハンドル

MATLAB（や C 等のプログラム言語）では、関数そのものを引数として、変数に代入して呼び出したり、引数として渡すことができる。例えば下の MATLAB コマンド列を見てみよう。

```
>> f = @sin;
>> f(1)
      ans = 0.8415...
>> f = @cos;
>> f(1)
      ans = 0.5403...
```

f という名前の関数定義は存在しない（f.m のようなファイルはない）のだが、上の最初の f(1) では sin(1) の値が、また２番目の f(1) では cos(1) の値がそれぞれ計算されている（単位はラジアン）。これは変数 f に、最初は sin 関数、２番目は cos 関数の「ハンドル」が代入され、f(1) によってそのハンドルの関数が実行されるからである。

関数そのものを呼び出すのではなく、ハンドルとして変数等に渡すには、上にあるように関数名の前に “@” をつける。関数ハンドルは、MATLAB の組込関数だけでなく、ユーザ定義の関数にも割り当てることができる。また「無名関数」を作ることもできる（詳細は省略）。関数ハンドルについては「help function_handle」を参照。

ユーザ定義の関数に関数ハンドルを渡し、実行する例を別紙の wave0.m に示す。wave0 では波形を生成する関数を関数ハンドルとして受け取り、その関数を呼び出して波形データを生成する（15 行目）。周期 2π の波形を生成する組込関数としては、sin, cos などの三角関数のほか、square（方形波）、sawtooth（鋸歯状波）などがあるので、それらを指定して wave0 を実行してみること。

2.9 作成した音データのグラフ表示

作成した音データが意図した波形になっているかを確認する方法の 1 つは、そのグラフを表示することである。

音データ y, サンプリングレート Fs の波形グラフを表示する一番簡単な方法は：

```
>> plot(y);
```

とすることだが、これでは全面塗り潰れたグラフにしかならないし、横軸の目盛りもデータ数で表示され、読み取りにくい。まず横軸を秒単位の目盛りにするには、次のように目盛りを入れた変数 t とともに plot を呼べばよい（t が音データ作成段階ですでに作ってあるなら、下の設定は不要）。

```
>> t = (0:(length(y)-1))/Fs;
>> plot(t, y);
```

最初の式の意味は、y の長さ（データ数）を n としたとき、0 から $n-1$ までの整数列を作成し、その値を Fs で割っている。それにより、データ 1 個の長さが 1/Fs（秒）になるわけである。

次に、表示範囲を調整するには axis を使う。例えば横軸を 0.0〜0.1 秒、縦軸を ± 1 の範囲で表示するには、下のようにすればよい。

```
>> axis([0.0 0.1 -1.0 1.0]);
```

なお公開 M-ファイルの tplot.m は、上記の処理を実現する関数である。これのソースを見て、どのような処理が行われているかを理解すること。

2.10 グラフ表示についての補足

グラフ（1 変数関数のグラフ）を描く上で、基本的な関連事項を以下に示す。（この項未完）

- 基本関数 plot

  グラフを描くための最も基本的な関数が plot である。

  plot(x, y) はデータ点列 x を横座標に、y を縦座標にとり、(x(i), y(i)) を順に折線でつないだグラフを表示する。

  x, y のデータは、必ずしも昇順にソートされていなくてもよい（下記の円や正多角形の作図も参照）。

  plot(y) は、y の長さを N (= length(y)) とするとき、plot(1:N, y) に等しい。

  plot は標準的には青色の折線グラフを表示するが、線の色・太さ・形状を指定したり、折線の代わりにマーカーでデータ点をプロットしたりもできる。詳しくは help plot 参照。

- 表示領域の指定、アスペクトの変更

  グラフの表示領域は MATLAB 側で自動的に設定されるが、ユーザ側で指定することもできる。上記のように：

```
>> axis([x1 x2 y1 y2]);
```

とすると、横軸方向は x1 から x2 まで、縦軸方向は y1 から y2 までの範囲が表示される。

また下記の円のグラフの作成などを行うと、そのままでは楕円形で表示されてしまう。これは表示領域が長方形で、さらに縦横比（アスペクト）が正しくないためである。

```
>> axis square;
```

とすれば表示領域は正方形になる。これと上記の表示範囲設定を合わせれば、正しいアスペクトでの表示ができる。

- グラフウィンドウの前面表示

```
>> shg;
```

とすれば、グラフウィンドウが最前面に表示される。グラフを作成したら合わせて shg と打つ習慣をつけておくとよい。

- グラフの重ね書き（重要）

plot を繰り返すと、前のグラフは消去されて新たなグラフが表示される。したがって複数のグラフを重ね書きしたい場合には、前のグラフを表示したまま上書きする指定をする必要がある。

```
>> hold on;
```

とすると上書き状態になり、以後の描画は前のものに上書きする。上書き状態を終了するには：

```
>> hold off;
```

とする。一連の描画が終わったら最後に hold off とすることを忘れずに。

- グラフ設定の集約

上記のようなグラフ作成上の様々な指定は、繰り返し用いるものなので、標準的な設定をプログラムとして保存しておくと便利である。私自身が使っている例（fgstart 関数）を下に示す。

```
%
% グラフ開始設定（関数形式）
%   fgstart(x1, x2, y1, y2, [axisflag])
%       表示範囲を [x1 x2] × [y1 y2] とする。
%       axisflag が指定され、0 のときは座標軸を表示しない。
%
function fgstart(x1, x2, y1, y2, axisflag)
if (nargin < 5)
    axisflag = 1;
end;
clf;
%plot([], []);
hold on;
axis square;
set(gca, 'colororder', [0 0 0]);
axis ([x1 x2 y1 y2]);
```

```
if (axisflag)
    plot([x1 x2], [0 0], 'k');
    plot([0 0], [y1 y2], 'k');
end;
line([x1 x2 x2], [y2 y2 y1]);   % patch: frame line
```

これを使って `fgstart(x1, x2, y1, y2)` などとすれば、表示範囲 `[x1 x2]` × `[y1 y2]` で座標軸のある正方形のグラフウィンドウが作成され、`hold on` 状態（上書き状態）で戻ってくる。

参考： `nargin` 変数

上の関数中では `nargin` という組込変数を参照している。`nargin` は関数呼び出しのときに渡された実引数の個数を値としている。これが仮引数の個数より少なければ、あとのほうの引数が省略されたことがわかり、プログラム側のデフォールト値を設定したりできる。

- グラフウィンドウの操作

  グラフが表示されているウィンドウ内では、表示の拡大・縮小・移動・回転（3D でしか意味がない）などにより表示内容を変更できる。

  「挿入」メニューからは、図形や文字列など、各種の追記を上書きできる。

  作成されたグラフは「ファイル」メニューから、各種の画像ファイル形式によって出力することができる。

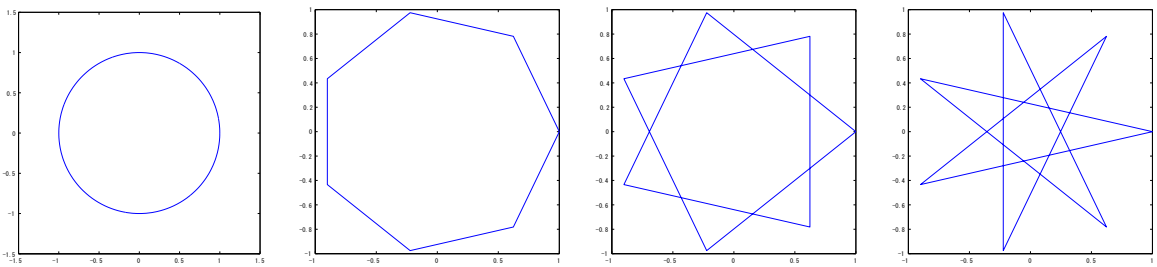
  これらの操作はコマンドウィンドウ内でコマンドを打ち込むことによっても実行でき、定型的な処理を繰り返すような場合にはそのほうがよい（M-ファイル内にも記述しておくなど）。

  例えば `text` 関数はグラフウィンドウへの文字列表示を、`print` 関数はグラフを画像ファイルに出力する。詳細はそれぞれの `help` を参照。

グラフ作成の演習課題

- $y = x^2$、$y = 1/x$、$y = \sin x$、$y = \sin x^2$ など、いろいろな関数を計算し、そのグラフを表示してみよ。
- $y = \sin t,\ y = \cos t$ のグラフを重ね書きしてみよ。両グラフは別の色（赤と青など）にすること。
- 円、さらには楕円を表示してみよ。
- 正 n 角形を表示してみよ。
- 星形（正多角形の対角線をいくつか置きに結んだもの）を表示してみよ。

下に円と、正７角形及びその星形の例を示す。



3 MATLAB で音（音響データ）を扱う

MATLAB 内では音響データも、単にディジタル化された数値列にすぎない。したがって通常の計算操作で作成・編集・入力／出力ができる。音響情報を扱うための機能として、コンピュータのオーディオ再生出力（内蔵スピーカー、ヘッドホン）から音データの再生、まだオーディオ入力から音響データの読み込みができる（Windows マシンのみ）。

ファイル入出力で扱えるデータの形式は WAV ファイル（`xxx.wav`）と AU ファイル（`xxx.au`）だけであり、他の形式（`mp3` 等）に対しては別途に入出力インタフェースを用意しなければならない。また MIDI データは標準ツールの範囲では扱えない。

> 補記: mp3 ファイルについては、最新版の MATLAB では入力ができる（`audioread`）。またサードパーティによるライブラリも存在する。

音響データを扱う関数・ツール類は、MATLAB の基本セットにあるものの他に、拡張パッケージである Signal Processing Toolbox, Wavelet Toolbox などの toolbox がある。ただし、音響・音楽データの扱いに必ずしも特化しているわけではない。

基本セットで使える音響関係関数は、`matlab/audiovideo` にまとめられている。コマンドウィンドウ内で：

```
>> help audiovideo
```

とすれば関数一覧が表示される。GUI ヘルプである helpwin からも同様に見ることができる。他にも関連関数（例えば FFT の実行）などはある。

また音響データ処理についての各種のデモプログラムもあり、それらの MATLAB ソースコードも自由に見ることができる。

3.1 基本事項

- MATLAB ではディジタル化した音響データを、−1〜+1 の範囲の実数値（浮動小数点数）として表わす。この範囲を超える値は切り捨てられる。
- 一定時間にわたる音響データは、この −1〜+1 の範囲の数値の１次元ベクトルとして表わす。各データの時間間隔は、サンプリングレートの逆数になる。

  例えばサンプリングレートが 8000Hz の場合、１つの数値は 1/8000 sec の長さに対応する。

  ステレオ（２チャンネル）の音響データの場合、２つの１次元ベクトルを並べた n 行 2 列の行列で表わす。
- MATLAB の文書等では、音響データベクトルを y（ないし Y）、サンプリングレートを Fs（ないし FS）、またデータのビット数を Bits（ないし NBITS）で表わされることが多い。
- サンプリングレートを省略した場合のデフォールト値は関数ごとに違うので注意。
  逆にいえば、デフォールトのサンプリングレートに頼るのは危険である。

3.2 データ例

以下のデータは、MATLAB にあらかじめ用意されている音響データサンプルである。

chirp　gong　handel　laughter　splat　train

いずれも：

```
>> load handel
```

のように load によって読み込むことができ、音響データは変数 `y` に、サンプリングレートは `Fs` に入れられる（それまであった値は上書きされるので注意）。いずれもモノラルである。

簡単なテストなどに利用するとよい。

3.3 基本的な関数

以下で [...] で囲ったのは省略してもいい引数を表わす。
デフォールト値や関数の詳細については help やマニュアルを参照。
なお wav... という名称の関数は Windows マシンだけでしか使えない。

- 音の再生（オーディオデバイスから）

```
sound(y [, Fs, NBITS])
wavplay(y [, Fs, 'async'|'sync'])
```

音響データ y（モノラルまたはステレオ）をマシンのオーディオデバイスで再生する。wavplay の async は再生中にコントロールが MATLAB に戻り、sync（デフォールト）は再生が終わるまでコントロールが戻らない（その代り、割り込みで再生を停止できる）。

sound を soundsc にすると、データ値が ±1 をはみ出した場合にはそれに収まるようにスケーリングを行う。

- 音の録音（オーディオデバイスから）

```
wavrecord(N [, Fs, Ch])
```

オーディオデバイスから N 個の音響データを入力する。
Ch はチャネル数で、1 ならモノラル、2 ならステレオ。

- WAV ファイルへの出力

```
wavwrite(y, [Fs, NBITS, ] ファイル名)
```

WAV ファイルに音響データを書き出す。NBITS は 8 ないし 16 を用いる。

- WAV ファイルからの入力

```
[y Fs NBITS] = wavread(ファイル名)
[y Fs NBITS] = wavread(ファイル名, N)
[y Fs NBITS] = wavread(ファイル名, [N1 N2])
[samples channels] = wavread(ファイル名, 'size')
```

WAV ファイルから音響データを読み込む。
音響データだけほしい場合には単に y = wavread('...'); と書けばよい。Fs や NBITS の値もほしい場合には [y Fs] = wavread('...'); のように [...] で囲って受け側変数を用意する（これは複数値を返す関数一般でも同じ）。

N を指定すると、最初の N 個のデータだけを読み込む。
[N1 N2] を指定すると、N1〜N2 番目のデータだけを読み込む。

'size' を指定すると、音響データは読み込まずに、ファイル中のデータ数（及びチャネル数）が返される。

- データ解析関数

音響データ解析のための関数は、Signal Processing Toolbox のもの、またデモプログラムなども加えると多数あるが、ここでは fft（高速フーリエ変換）にだけ触れておく。基本的なデータ解析関数については matlab/datafun を参照。

```
fft(y [, N])
ifft(y [, N])
```

（離散複素）高速フーリエ変換を行う（ifft は逆変換）。

結果は複素フーリエ変換の係数 c_n のベクトルとして返ってくるため、このままではあまり使えない。したがって結果を自分でさらに加工する必要がある。

普通ほしいのはパワー値 $|c_n|^2$ だから、まず結果の絶対値をとって 2 乗する。また実関数のフーリエ変換のため、共役性 $c_{-n} = \overline{c_n}$（複素共役）となるため、実質的な情報は係数ベクトルの半分しかない。そこで前半部分だけを取り出せばよい。

N を指定すると、N 点のデータに対して変換を行う。
データ点数は 2^n の形（2 のべき乗）であることが望ましいが、それ以外の値を指定しても適宜補填されて処理される（精度は一般に落ちる）。

端点誤差を避けるためには、hanning, hamming 等の窓関数を y に乗じてから FFT を行う。

サンプリングレート Fs の データ N 点に FFT を施した場合、結果は 1 点あたり Fs/N (Hz) の周波数幅になる。

4 MATLAB プログラムの作成・編集

MATLAB プログラム（実行系列）はファイルとして保存し、読み出して実行することができる。MATLAB プログラムファイルには大きく分けて、M-形式（script ファイル）と関数形式とがある。

- M-形式、関数形式のいずれも、ファイル名には '.m' という拡張子をつける。読み込むときにはそのファイル名（'.m' は不要）を、普通の関数名と同じように打てばよい。逆にいえば、組込関数（sin など）と同名のファイルは作れない、あるいは作るべきではない。
- ファイル読み込みは、（パス名を指定しなければ）「現在ディレクトリ」から行う。
- ディレクトリの表示は dir（あるいは ls）コマンドにより、現在ディレクトリの変更は、コマンドウィンドウ上部のディレクトリ表示ウィンドウで指定するか、cd コマンドにより行う。

- MATLAB 内で「>> edit ファイル名」とすれば、そのファイルを編集するウィンドウが開く。
  .m ファイルはテキストファイルなので、普通のテキストエディタでも編集できる。
- 行中に % があると、それ以降はコメントとみなされる。

- M-形式のファイルは、コマンドラインで MATLAB 実行文を打つのと同じ要領でファイルに実行文列を打ち込めばよい。

  各変数は、すべてグローバルスコープの変数として扱われる。

- 関数形式のファイルは、先頭に：

  ```
  function [r1 r2 ...] = fn(a1, a2, ...)
  ```

  のように関数宣言を入れる。

  fn は関数名で、ファイル名と同じにする（同じでなければ正常に実行できない）。a1, a2, ... は関数の仮引数、r1 r2 ... は返り値の変数名である。返り値は、関数の中でその変数に代入すれば自動的に設定されるので、C のように return 文などを用いる必要はない。

関数形式では、M-形式と違って変数はすべてローカルスコープである。したがってグローバル変数の値に影響を及ぼさない一方、グローバル変数の値を参照するには、引数で値を渡すか、global 宣言を行う必要がある（詳細略）。

関数形式ファイルの中では下請け関数を上と同様の形式で定義できる。下請け関数は外部からは直接参照することはできない。関数の終わり部分を指定する記述はなく、次の function が来る直前までが関数定義本文になる。

逆に M-形式では下請け関数を定義できない（なんで〜〜？）。

5 プログラム例

下に、正弦波データを生成する（関数）プログラム例 ssin0.m を示す。
このように本文３行ほどで定義できてしまう反面、このままでは関数呼び出しですべての引数値を与えなければならない、その一方で振幅など自由に設定できないという不自由さがある。

```
%
% ssin0.m: 正弦波の生成（基本形）
%
%   f    : 周波数
%   tlen: 時間長（秒）
%   Fs   : サンプリングレート

function [y Fs] = ssin0(f, tlen, Fs)
dt = 1/Fs;                    % １サンプルの時間長
t = 0:dt:tlen;                % サンプル時間列
y = sin(2*pi*f.*t);
```

上を改良したのが ssyn.m である。ssyn.m については講義資料の wave2.pdf にも掲げてあるが、次ページ以降ににその説明を再掲しておく。

5.1 ssyn.m の使い方（wave2.pdf より転載）

ssyn.m（http://www.slis.tsukuba.ac.jp/~hiraga/music/tsukubaonly/wav.shtml）は与えられた周波数・振幅・位相を持つ正弦波を合成し、演奏するための Matlab 関数プログラムである。生成した音データは、tplot.m によって波形グラフを表示できる。Matlab の勉強のためには、上記ページにある他のプログラムも適宜参照するとよい。使用するプログラムは、上記 Web ページからダウンロードして、Matlab 実行時に使用する適当なディレクトリにコピーしておけばよい。ssyn.m のソースコードは p.17 に掲げておく。以下文中で言及する行番号はこのソースのものである。なお Web 掲載プログラムは随時改訂する可能性があるので、掲載版とは違いが生じることはありうる。

ssyn.m は正弦波を合成した音を作成するので、音量・音高が時間的に変化するような音は作りにくい（ただし後述参照）。ただ、フーリエ級数のように基本周波数の倍音だけでなく、いろいろな周波数成分を組み合わせられるという点では自由度が高い。

基本的な使い方

- `help ssyn;`

  ssyn の使い方の説明。実際には、ソースコード上端にあるコメント部分（1〜13 行）を表示するだけ。引数の種類や順番を見るには有効。

- `y = ssyn(440, 1);`

  ssyn は関数呼び出しで実行される。第 1 引数は周波数（スカラーまたはベクトル：単位は Hz）、第 2 引数は音長（単位：秒）である。上の例では、440 Hz の正弦波の音 1 秒分の波形データが作成され、それが変数 y に代入される。末尾のセミコロンを忘れると、数万点の全データが画面表示されてしまうので注意。

  引数の値は整数値、小数点付きの実数値のいずれでもよい。また `Do = 261.6;` のように変数に代入しておき、`ssyn(Do, 1)` のようにその変数を引数にしてもよい。

  第 2 引数の音長は省略可能で、省略した場合には 1 秒になる（デフォールト値）。またデフォールトのサンプリングレートは 44100 Hz である（CD のサンプリングレートと同じ）。

  上は音データを返すだけで、音は鳴らさない。音を鳴らすには下記のように wavplay などを使う。

- `[y Fs] = ssyn(440, 1);  wavplay(y, Fs);`

  上と同じだが、音データを変数 y に代入するだけでなく、第 2 の値としてサンプリングレートが変数 Fs に代入される（実際には第 3 の値として、データ点数も返される。これは `length(y)` に等しい）。

  このあとの `wavplay(y, Fs);` によって音データをスピーカーなどのオーディオ出力機器で鳴らすことができる。もちろん `wavplay(y, 44100);` のようにサンプリングレートを値を直接指定してもよい。

  注： wavplay も第 2 引数のサンプリングレートを省略できるが、省略時の値は 11025 Hz なので注意。つまり 44100 Hz の音を `wavplay(y);` のように鳴らすと、実際の音より 2 オクターブ低く、長さが 4 倍の音が鳴る。

- `y = ssyn([440 550 660], 2);`

  ベクトル（＝1 次元配列）により複数の周波数を指定した例。これらの周波数の正弦波で、振幅（　音の強さ）が同じ 3 音を足し合わせた（合成した）音データが作成される。上では周波数比 4:5:6 の、長 3 和音（「ドミソ」の和音）の音が作られる。

  ベクトルの一番基本的な作り方は、上のように数値を [...] で囲うことである。より高度な使い方については後述。合成された音データは、数値（周波数値）の順番を変えても、合成される音データは同じである。なお最初の例は、`ssyn(440, 1)` と書いても、`ssyn([440], 1)` のように 1 要素のベクトルとして書いてもよい（Matlab では同一視される）。

  振幅（y の値）が Matlab の上限の ±1 をはみ出す場合には、その範囲に収まるように自動調整される。これを「正規化 (normalize)」という（ソースコード 70–72 行目参照）。

キーワードパラメタ

周波数（ベクトル）、音長以外の引数は「キーワードパラメタ」として指定する。キーワードパラメタとは、引数の種類を表すキーワード（'...'（シングルクォート）で囲った文字列）と、その引数の値とを並べて指定する方法である。キーワードパラメタは指定しても省略してもよく、またどのような順番で指定してもよい。ただし、キーワードの値は必ずそのキーワードの直後に置かなければならない。

キーワードパラメタを指定する場合、音長（第２引数）は省略できない。

> このような引数の指定方法を実現する Matlab の機能が変数 `nargin`, `varargin` である。`nargin` は、関数呼び出し時に渡された引数の個数を値とする特別な変数である。したがって引数が省略されているかは、関数の仮引数の個数と `nargin` とを比較することでわかる（ソース 23–25 行参照）。
>
> 一方、`varargin` は不定個の引数を１つのセル配列にまとめた特別な変数である。したがってその値を順に調べていくことで、どういう引数値が渡されたかを知ることができる。ユーザの側から見れば、引数を省略したり、順番を自由に並べたりできるというメリットがある。`varargin` に渡されたセル配列の処理方法についてはソースコード 27–43 行参照。普通の配列と異なり、セル配列は `varargin{i}` のように、添字を (...) ではなく、{...} で囲う点に注意。

`ssyn` のキーワードパラメタは、'Fs'（サンプリングレート）、'A'（振幅（ベクトル））、'ph'（位相（ベクトル））、'play'（音を鳴らす：パラメタ値なし）の４つがある。ソース 30–41 行の `switch ... case` 文にあるように、例えば 'A' は 'Amp', 'amp' などと書いてもよい。

- `ssyn(440, 1, 'play');`

  440 Hz の正弦波音を１秒鳴らす。これまでの例のように音データを変数 `y` に代入していないのは、単に音を鳴らせばよく、データそのものは保存しておく必要がないことを意味している。

- `y = ssyn([440 550 660], 2, 'A', 0.25);`

  振幅の一律指定。振幅のデフォールト値は最大値の 1 である。上は長３和音の３音のいずれも、振幅 0.25、つまり最大値の 1/4 の値になる。

- `y = ssyn([100 200 300 400], 1, 'A', [1 1/2 1/3 1/4]);`

  振幅ベクトルを使って、各周波数成分の振幅を個別に指定した例。上は基本周波数 100 Hz の音の第４倍音までを、振幅（フーリエ係数値）を周波数に反比例させて合成する（つまり n 倍音の振幅は $1/n$）。これは鋸歯状波の近似にあたる。ベクトルの成分中でも、`1/2` のような数式や変数を書いてかまわない。

  ベクトルの成分を羅列するのは、成分数が多い場合には面倒だが、この例はもっと簡単な形に書ける（後述）。

- `y = ssyn([1 2 3], 1, 'ph', [0 a b]);`

  位相を指定した例。第１成分の位相は 0、第２・３成分の位相は変数 `a`, `b` によって与えられている。p.20 の図のデータは、上を用いて作成したことに相当する。

- `y = ssyn([440 440], 1, 'A', [1/2 sqrt(3)/2], 'ph', [0 pi/2]);`

  上の例のように周波数に同じ値を指定してもかまわない（あまり必要性はないが）。また複数のキーワードパラメタを同時に指定してもよく、その順番も自由である。`sqrt` は平方根を計算する関数。位相が `pi/2` というのは、sin を cos にするということだから、上は結局：

  $$y = \frac{1}{2}\sin\omega t + \frac{\sqrt{3}}{2}\cos\omega t = \sin(\omega t + \tfrac{\pi}{3}) \quad (\omega = 2\pi \times 440)$$

  という正弦波を合成したことにあたる。

- `[y Fs] = ssyn(440, 1, 'Fs', 11025);`

  サンプリングレートを `wavplay` デフォールトの 11025 に指定。

ベクトルの書法

ベクトルは [100 200 300] のように成分を並べて [...] で囲うのが基本である（成分間に [100, 200, 300] のようにカンマを入れてもかまわない）。しかしベクトルを作成する式や、ベクトル演算を使えば、複雑なベクトルを簡単に作成することができる。詳細については Matlab 関連資料も参照。

- a = zeros(1, N);　長さ N のゼロベクトルを作成する。
- a = ones(1, N);　長さ N で成分がすべて 1 のベクトルを作成する。
  下記のようにこれを定数倍すれば、任意の値を羅列したベクトルを作ることができる。
- 階差列（コロン演算子） a:b, a:d:b

  a:b は a から始まり、階差 1 で b 以下の数列を作成する。　例： 1:5 → [1 2 3 4 5]

  a:d:b は a から始まり b に至る、階差 d の数列を作成する。
  d > 0 なら上昇列、d < 0 なら下降列になる。

  - 1:2:9 → [1 3 5 7 9]　（奇数列）
  - 1.1:0.5:2.3 → [1.1 0.6 2.1]　（2.3 以下の値まで作られる）
  - 10:-3:0 → [10 7 4 1]　（下降列になる）
  - 0:pi/180:2*pi　$0, 2\pi$ の間を 360 等分する（つまり 1 度単位に分割）。
    両端が含まれるので、データ数は 361 になることに注意。
- ベクトルへの演算

  個別に説明するのは長くなるので、例示するにとどめる。

  - [1 2 3] + 10 → [11 12 13]　（成分への和）
  - [1 2 3] + [4 5 6] → [5 7 9]　（ベクトルの和）
  - [1 2 3]*100 → [100 200 300]　（成分への積）
  - [1 2 3].* [1 2 3] → [1 4 9]　（成分ごとの積：演算子 .* に注意）
  - 1 ./ [1 2 3] → [1 1/2 1/3]　（成分ごとの逆数：演算子 ./ に注意）

  以下 a = [1 2 3], b = [4 5 6] とする。

  - [a b] → [1 2 3 4 5 6]　（ベクトルの連結）
  - a = [a 4] → a = [1 2 3 4]　（要素を増やして連結）
  - b(2:3) → [5 6]　（部分列の抽出）

  上の例にあるように、Matlab のベクトルは C の配列とは異なり、配列の大きさを大きくする、一部分を取り出すといった操作が自由にできる。また a = [1 2 3] に対し、a(6) = 6 とすると、a の大きさは 6 にまで広げられ、間には 0 を埋めて a: [1 2 3 0 0 6] となる。

  ただし、配列の大きさを変えるとその都度メモリーコピーが生じるので、数万点以上の巨大配列については、大きさを動的に変えることはせず、最初に十分な大きさを（zeros などを使って）割り当てておくことが望ましい。この点では C などと同様である。

これらの演算を組み合わせると、様々なベクトルが簡単に書ける。例えば（レポート問題にかぶるが）基本周波数が 100 Hz で $b_n = \frac{1}{n}$ である音の合成（1〜20 成分まで）は：

```
NN = 1:20;
y = ssyn(NN*100, 1, 'A', 1./NN);
```

のように簡単に書くことができる。
また基本周波数 100 Hz、5 オクターブのユニゾンは、ssyn(100 * 2.^(0:4), 1) によって作ることができる（なぜか？）。

応用例：減衰音の作成

先に、ssyn は正弦波合成プログラムであるため、音がだんだん小さくなるような減衰音のような、時間変化を伴う音は作れないと書いた。しかしここらは工夫次第でいろいろやりようはある。

そこで長さが 1 秒で、段々小さくなるような音を作ってみよう。それにはまず、$t = 0$ では 1 で、以下単調減少して $t = 1$ で 0 になるような、長周期の関数を用意する。具体的には、周期 4 秒の cos 関数 $\cos \frac{2\pi}{4} t$ が簡単である。これに周波数 f（例えば $f = 440$）の正弦波を掛け合わせる：

$$y(t) = \cos\left(\frac{2\pi}{4}t\right) \sin 2\pi f t$$

このままでは正弦波の和の形をしていないので、積和の公式（p.1 (1.3)）により分解する：

$$y(t) = \frac{1}{2} \sin 2\pi \left(f + \frac{1}{4}\right) t + \frac{1}{2} \sin 2\pi \left(f - \frac{1}{4}\right) t$$

これにより、２つの正弦波の和の形になったので、

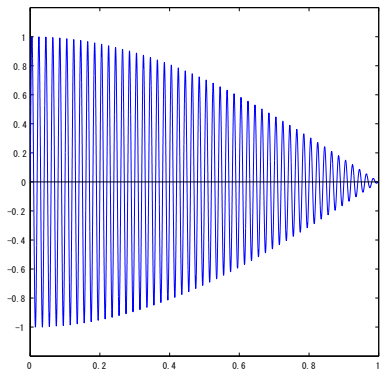
```
f = 440;
y = ssyn([f+1/4 f-1/4], 1, 'A', [1 1]/2);
```

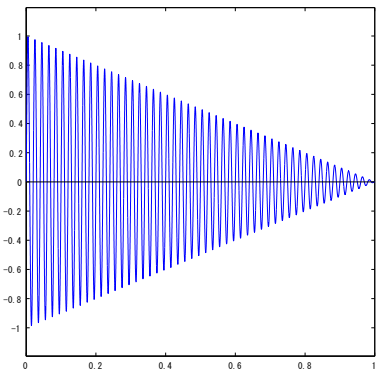
のようにすればよい。これは周波数差が 0.5 Hz、周期 4 秒で生じる「うなり」の 1/4 周期分にあたる。これを tplot でグラフにしたものを下左図に示す。ただし、細かい振動が段々小さくなっている様子がわかるように、図では f=50 にしてある。

一方、ssyn だけの使用にこだわらないなら、正弦波のデータに後から減衰をかけることもできる。例えば 1 秒間に直線的に減衰する音を作るなら、$\sin 2\pi f t$ に $y = 1 - t$ を掛ければいいから：

```
y = ssyn(f, 1);
n = length(y)-1;
y = y .* (n:-1:0)/n
```

のようにすればよい。これを tplot でグラフにしたものを下左図に示す。先ほど同様、f=50 にしてある。
これらを実行して、実際に減衰する音になっていることを確認してほしい。





tplot の使い方

上でも使ったが、tplot は音データのグラフを、横軸を時間軸（単位は秒）として指定した範囲で表示する。以下 y は音データ、Fs はサンプリングレートである。

- `tplot(y, Fs);`　　全データのグラフを表示。Fs を省略すると 44100 Hz になる。
- `tplot(y, Fs, t);`　　時刻 0 から t までの部分を表示。
- `tplot(y, Fs, t1, t2);`　　時刻 t1 から t2 までの部分を表示。
  ただし、t2 = 0 のときは、時刻 t1 から音の最後までの部分を表示する。

グラフの縮尺は自動的に設定される。

```
 1: %
 2: % ssyn.m: 正弦波の生成・合成 (09/11/13; 12/01/26 YH)
 3: %
 4: %   [y Fs Ndata] = ssyn(f, tlen [, options])
 5: %       f    : 周波数
 6: %       tlen: 時間長（秒）
 7: %       他のパラメタはキーワードパラメタとして指定する。
 8: %       f がベクトルのときは、それらの周波数の正弦波を合成する。
 9: %       キーワードパラメタ
10: %           'Fs': サンプリングレート（デフォールト 44100）
11: %           'A' : 振幅（ベクトルの場合、周波数ベクトルの各成分の振幅）
12: %           'ph': 位相（ベクトルの場合、周波数ベクトルの各成分の位相）
13: %           'play': （パラメタなし）: 音を鳴らす。
14:
15: function [y Fs Ndata] = ssyn(f, tlen, varargin)
16: %
17: % 以下、デフォールト値の設定
18: %
19: Fs = 44100;                % サンプリングレート
20: A = 1.0;                   % 振幅
21: ph = 0.0;                  % 位相
22: play = 0;                  % 演奏（デフォールトは鳴らさない）
23: if (nargin < 2)            % デフォールト時間長：1秒
24:     tlen = 1.0;
25: end;
26:
27: i = 1;
28: vl = length(varargin);
29: while (i <= vl)
30:     switch (varargin{i})    % 注意：エラーチェックはしていない
31:         case {'Fs', 'fs', 'FS'}                              % サンプリングレート
32:             i = i + 1; Fs = varargin{i};
33:         case {'A', 'Amp', 'amp', 'Amplitude', 'amplitude'}  % 振幅
34:             i = i + 1; A = varargin{i};
35:         case {'ph', 'Ph', 'Phase', 'phase'}                  % 位相
36:             i = i + 1; ph = varargin{i};
37:         case {'Play', 'play'}                                % 音を鳴らす
38:             play = 1;
39:         otherwise
40:             error('error in argument %d\n', i+2);
41:     end;
42:     i = i + 1;
43: end;
44:
45: dt = 1/Fs;                                  % 1サンプルの時間長
46: if (mod(tlen, dt) ~= 0)
47:     t = 0:dt:tlen;                         % サンプル時間列
48: else
49:     t = 0:dt:(tlen-dt);                    % tlen が dt で割り切れる場合には最後の1点を除く
50: end;
51: Ndata = length(t);
52:
53: y = A(1) * sin(2*pi*f(1).*t + ph(1));      % 最初の成分
54: if (length(f) > 1)
55:     if (length(ph) < length(f))            % 周波数ベクトルに対応した位相ベクトルの設定
56:         for i = length(ph)+1:length(f);
57:             ph(i) = ph(1);
58:         end;
59:     end;
60:     if (length(A) < length(f))            % 周波数ベクトルに対応した振幅ベクトルの設定
61:         for i = length(A)+1:length(f);
62:             A(i) = A(1);
63:         end;
64:     end;
65:     for i = 2:length(f);                   % 2番目以降の成分の合成
66:         y = y + A(i) * sin(2*pi*f(i).*t + ph(i));
67:     end;
68: end;
69:
70: % y = normalize(y);                        % 正規化：以下でインライン処理
71: ymax = max(abs(y));
72: if (ymax > 1); y = y / ymax; end;
73:
74: if (play ~= 0); wavplay(y, Fs); end;       % 音の演奏
```

5.2 実例・演習（本日の課題）

5.2.1 MATLAB プログラムの編集・実行

- 上の ssin0 を ssin0.m という M-ファイルに作成し、実行せよ。

```
>> f = 440;
>> Fs = 8000;
>> y = ssin0(f, 1, Fs);
>> wavplay(y, Fs);
```

のようなコマンド列により、実際に音が発音されることを確認せよ。

- 上で f, Fs の値を変えるとどうなるか。
また wavplay 実行時に Fs の値を変えるとどうなるか。

- できれば f の値をいろいろ変えて、音階の音（ドレミファソラシド）を鳴らしてみよ。

- ssyn のプログラムも打ち込んで実行してみよ。ssin0 との違いを比較せよ。

5.2.2 音の重ね合わせ

- 異なる高さの音を重ねるには、単純にその波形データを加えればよい。

```
>> f1 = ...;
>> f2 = ...;
>> Fs = 8000;
>> r = 0.5;
>> y = r * ssin0(f1, 1, Fs) + (1-r) * ssin0(f2, 1, Fs);
>> wavplay(y, Fs);
```

のように２つ、あるいはそれ以上の音を、周波数 f1, f2 をいろいろ変えて重ね合わせてみよ。また r の値を変えるとどうなるか。

5.2.3 正規化

重ね合わせにより音を作っていくと、値が ± 1 の限界を超えてしまう場合がある。soundsc 関数で演奏すれば出力時に自動調整されるが、これを自分のプログラムでやってみよう。

一番簡単な考え方としては、（絶対値の）最大値が 1 以下になるように全体を割ればよい。次の normalize 関数はそれを行うものである。

```
function [y maxA] = normalize(y)
maxA = max(abs(y));
y = y ./ maxA;
```

ただし、y の値がすべて 0 だと maxA も 0 になり、0 割りエラーになる。またこれだと小さな音でも最大振幅 1 になってしまう。

- 最大振幅がもともと 1 より小さければ、そのまま出力するように上を改良せよ。
（ヒント： maxA が 1 より大きいかどうかを、if 文を使って判断する。）

5.2.4 うなり

周波数 f の正弦波に対し、それから少しだけずれた $f+\Delta f$ の音を重ねると「うなり」が生じる。これは：

$$\sin 2\pi ft + \sin 2\pi(f+\Delta f)t = 2\sin 2\pi\left(f+\frac{\Delta f}{2}\right)t\cos 2\pi\frac{\Delta f}{2}t$$

となって、$\Delta f \ll f$ なら cos の項の時間変化が sin の項の時間変化に対して極めてゆっくりとなり、周波数 $f+\dfrac{\Delta f}{2}$ の正弦波に対し、$2\cos 2\pi\dfrac{\Delta f}{2}t$ が（時間的に変化する）振幅のように働くからである。

これは振幅変調（Amplitude Modulation: AM）の一種と言える。

- 上を実現する MATLAB プログラムを作成し、Δf をいろいろ変えて試してみよ。