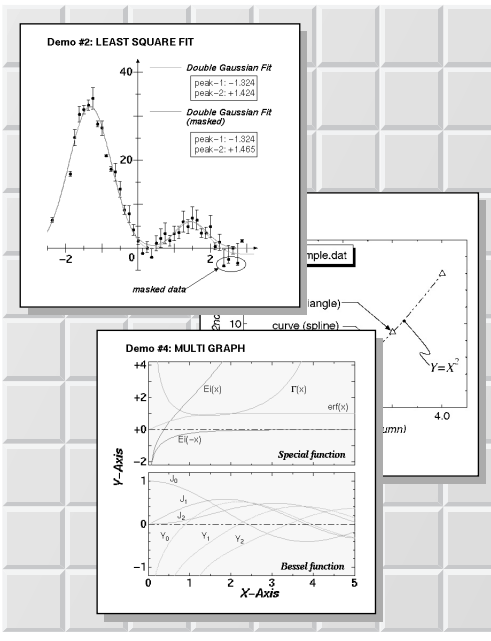
Ngraph [10] p89-96 99.08.19 17:38 ページ 89



短期集中連載

Ngraphでグラフの達人になろう

最終回

アドインの利用と高速フーリエ変換

石坂　智

Ngraphにはアドインと呼ばれる外部プログラムを実行する機能が備わっている。今回は高速フーリエ変換（FFT）を例として、作成方法とそれを用いたグラフ作成方法を解説する。また今回、LibCD Vol.67 Disk 2の/rensai/Ngraphには、田中周氏作成によるNgraphのRPMパッケージを収録しているので、Red Hat Linux系のユーザーはこれを利用するといいだろう。

誤差棒、矢印

前回、学生実験の実例をあげながらグラフの作成方法を解説したが、そこでは（x、y）の組のデータだけを扱ってきた。Ngraphでは（x、y）とは異なる２つのデータ形式も扱えるので、アドインの解説の前に補足しておく。実験の場合、測定データに誤差が含まれていて、その誤差の範囲もグラフに表示したいことがある。たとえば、データが（x、y±err）と、yに±errだけ誤差がある場合には、

```
x y +err -err
```

のように、１行に４つの列からなるデータ・ファイルを作成し、プロット・タイプを「errorbar_y」にする。**図１**はこのようにして誤差棒を表示したグラフ例である。同様に、xに±errだけ誤差がある場合には、

```
x +err -err y
```

のようなデータ・ファイルを作り、プロット・タイプを「errorbar_x」にし、「(X)カラム」を“１”に「(Y)カラム」を“３”にする。

この誤差棒のほかにもう１つ、データ・ファイルの形式が通常とは異なり、プロット・タイプが「diagonal」、「arrow」、「rectangle」の場合がある。これらのプロット・タイプでは(x1,y1)－(x2,y2)に、それぞれ対角線、矢印、長方形を書けるようになっているので、データ・ファイルの形式は、各行が、

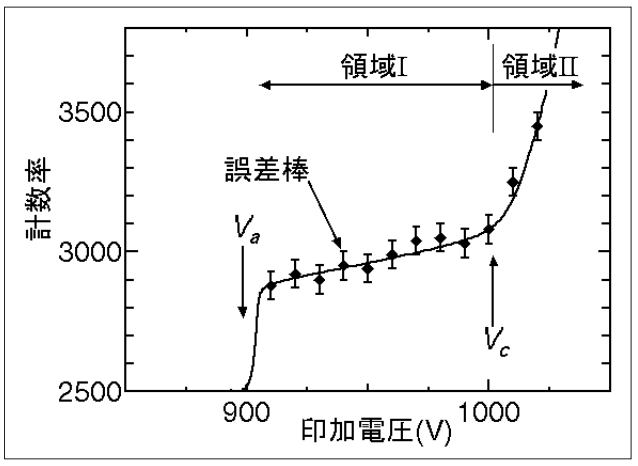
```
x1 y1 x2 y2
```

となるようにする。また「(X)カラム」を“１”に「(Y)カラム」を“３”にする。

アドインとは

これまでNgraphに備わっている機能だけを使って、グラフの作成方法を一通り解説してきたが、用意された機能だけでは満足できない場合もあるだろう。また、グラフ作成でいつも同じデータ処理が必要な場合、その処理が自動化できれば、より便利になるだろう。そのようなときにアドインを

図１　誤差棒を表示したグラフ



作成すれば、ユーザー自身でNgraphに疑似的な機能を追加できる。

アドインとは基本的に外部プログラムのことであり、Ngraphは現在の設定を取得/変更できる仕組みを用意しているので、外部プログラムをあたかもNgraphの１つの機能のようにして利用できる。この仕組みはDOS版Ngraphからの大きな特徴の１つであった。しかし、DOS版ではメモリ上のデータをプログラム越しに直接アクセスしていたので、システムに依存し汎用性にも欠けていた。そこで現在のVersion 6.0では、後述するスクリプトのインタープリタを内蔵し、スクリプトで外部プログラムとNgraphのインタフェースを取るように変更している。

また、このスクリプトをグラフの保存形式としても使っている。つまり、グラフを保存したファイル（拡張子ngpを持つファイル）がスクリプトそのものなのである。さらに、Ngraphの初期化やコマンド・ライン・オプションの解析もスクリプトで行っている。

付属アドインの使い方

Ngraphにはあらかじめいくつかのアドインが付属している。これらはTcl/Tk（wishのスクリプト）で書かれているので、使用する前にTcl/Tkをインストールしてほしい。付属アドインは、メニューから「グラフ」→「アドイン」で一覧が表示されるので（**図２**）、このリスト・ボックスの中から選ぶだけで利用できる。付属アドインの中で、利用頻度の高そうなものを解説しよう。

APPEND

「グラフ」→「開く」で保存してあるグラフを読み込んだ場合、現在作成しているグラフは消去されてしまう。現在作成しているグラフはそのままで、そこに保存してあるグラフを追加したい場合にAPPENDアドインを使う。使い方は、**図２**で「APPEND」を選ぶとファイル選択のダイアログ・ボックスが開くので、追加したいグラフのファイルを指定する。前回、複数のグラフを作成する方法を解説したが、このAPPENDを使っても複数のグラフを作成できる。

凡例の自動生成LEGEND

データの凡例を自動生成するにはLEGENDアドインを使う。**図２**で「LEGEND」を選ぶと**図３**のダイアログ・ボックスが表示される。下側のリスト・ボックスにはNgraphで開いたデータ・ファイルの一覧が表示されているので、ダブル・クリックしてデータを説明するテキストを入力する。デフォルトはファイル名になっている。また、「Frame」をチェックすると、凡例全体が影の付いた枠で囲まれるようになる。LEGENDアドインで作成した凡例が**図４**である。

第１回で、データを線とマークの両方でプロットするには、同じデータ・ファイルを２度開いて、片方のプロット・タイプ

図２　メニューの［グラフ］→［アドイン］で表示されるダイアログ・ボックス

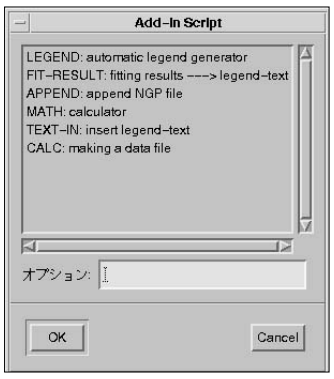
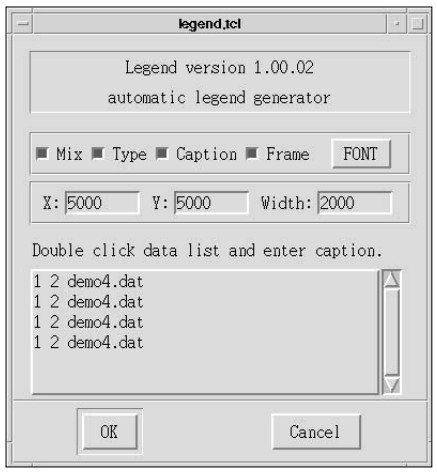


図３　LEGENDアドインのダイアログ・ボックス



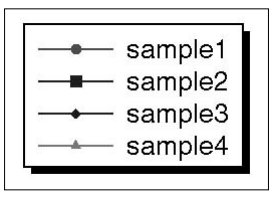


図４　LEGENDアドインで作成した凡例

を「line」に、もう片方を「mark」にすればよいと解説したが、そのような場合に**図2**で「Mix」をチェックすると、2つのデータ・ファイルで1つの凡例を作成するようになる（**図4**）。これまでの解説で紹介してきたグラフの凡例も、LEGENDアドインで作成されたものが多い。

フィット結果の凡例作成FIT-RESULT

フィット結果をグラフに凡例として入れるには、FIT-RESULTアドインを使う。**図2**で「FIT-RESULT」を選ぶと**図5**のダイアログ・ボックスが表示される。中程のリスト・ボックスには、プロット・タイプが「fit」になっているデータ・ファイルの一覧が表示されているので、この中から選べばよい。「%00」～「%09」がフィット結果で、その「Caption」には係数の説明を入力する。「accuracy」でフィット結果の有効桁数を指定する。

また、「Expand」で凡例を動的に変更するか否かを指定できる。ここをチェックしていると静的に作成し、現在保持されているフィット結果（直前の描画時のフィット結果）を使って凡例テキストが作成される。この場合、フィット結果が変わったら、もう一度FIT-RESULTアドインを起動して凡例を作成し直す必要がある。

チェックしていない場合には、数値が記述されるのではなく、たとえば、「1番目のフィット結果の係数%01を表示する」といった形式の凡例テキストが作成される。具体的にはテキストに「%{fit:1:%01}」が入力される（実際は有効桁によるフォーマッタが入るので、もう少し複雑だが）。後でデータをマスクするなどしてフィットの結果が変わった場合、凡例も自動的に変更され、いつでもグラフの画面上で最新の結果を確認できるので便利である。またグラフのひな型を作る場合にも便利だろう。このFIT-RESULTアドインで作成した凡例が**図6**である。

任意関数の表示CALC

任意関数のグラフを作成するのがCALCアドインである。実験データのグラフを作成していると、そこに理論曲線を重ねて表示したい場合もあるだろう。そのようなときに便利なものだ。**図2**で「CALC」を選ぶと**図7**のダイアログ・ボックスが表示されるので、そこで次の内容を指定する。

Output	作成するデータのファイル名
Formula	計算したい関数式
Minimum、Maximum	引数Xの範囲
Include min、Include max	端点を含ませるかどうか
Division	引数の範囲を何分割するか

図5　FIT-RESULTアドインのダイアログ・ボックス

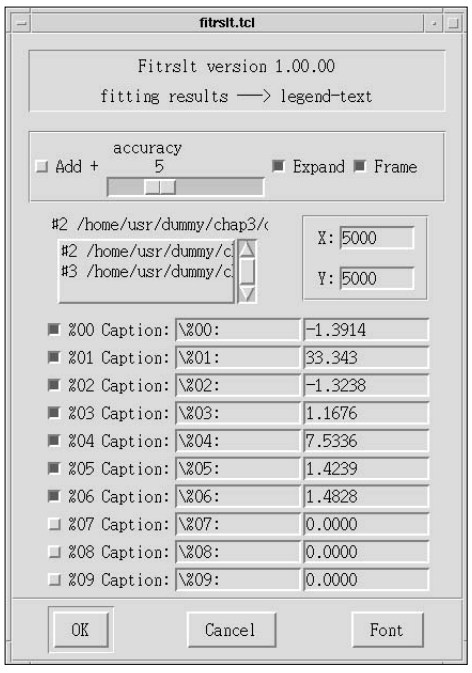


図6　FIT-RESULTアドインで作成した凡例

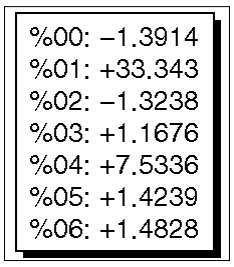


図7　CALCアドインのダイアログ・ボックス

これらのパラメータは出力ファイルにコメントの形で記述されているので、CALCアドインで作成したデータ・ファイルを「Output」に指定して「LOAD」ボタンを押せば、以前の設定を読み込める。「Open as data」がチェックされていれば、作成されたデータ・ファイルが描画ファイルとして自動的に開かれる。

ここで注意してほしいのは、CALCアドインは「Mininum」から「Maximum」までを等間隔に分割したデータ・ファイルを作成するだけで、実際に関数式を計算して値を出す訳ではないことだ。関数式の計算はNgraph本体の数式変換機能によってグラフ描画時に行われ、「Formula」には、Ngraphの数式変換の関数がすべて使える。

関数式の計算結果をファイルに出力したい場合には、CALCアドインの実行後に「出力」→「データファイル」として、グラフの値をファイルに出力すればよい(後述)。

高速フーリエ変換(FFT)アドイン

今度は高速フーリエ変換(FFT)による振動波形解析を例にして、アドインを使ってデータ処理を自動化する方法を解説しよう。

インストール

FFTアドインのソースaddin-fft-1.00-src.tar.gzをLibCD Vol.67 Disk 2の/rensai/Ngraphに収録している。まず、このアドインのソースを、

```
# tar xvzf addin-fft-1.00-src.tar.gz
```

のようにして適当なディレクトリに展開する。ここで展開される、fft.cはFFTを行うCのソース・ファイル、fft.tclは簡単なGUIを提供するTcl/Tk(wish)スクリプト、fft.nscはFFTアドインと、Ngraphとのインタフェースを取るNgraphスクリプトである。

展開し終わったら、fft.cをコンパイルする。

```
# cc -o fft fft.c -lm
```

続いて、作成されたバイナリfftと、fft.tcl、fft.nscをNgraphがインストールされているディレクトリにコピーする。

```
# cp fft fft.tcl fft.nsc /usr/local/lib/Ngraph
```

以上で関連ファイルのインストールは終了だ。最後にFFTアドインをNgraphに登録する作業を行う。メニューから「設定」→「アドインスクリプト」で表示されるダイアログ・ボックス(**図8**)で「追加」ボタンを押す。「名称」には、たとえば、

```
FFT: Fast Fourier Transform
```

を入力する。「スクリプト」にはインストールしたNgraphスクリプト

```
/usr/local/lib/Ngraph/fft.nsc
```

を指定する(**図9**)。続いて、次回の起動でも設定が有効になるように、メニューから「設定」→「初期状態としてセーブ(各種設定)」で表示されるダイアログ・ボックスで「アドイン設定」ボタンをチェックして「OK」ボタンを押す。これで登録作業は完了だ。メニューから「グラフ」→「アドイン」で表示されるダイアログ・ボックス(**図1**)にはFFTアドインが追加されているはずなので、確認してほしい。

図8 [設定]－[アドインスクリプト]メニューで表示されるダイアログ・ボックス

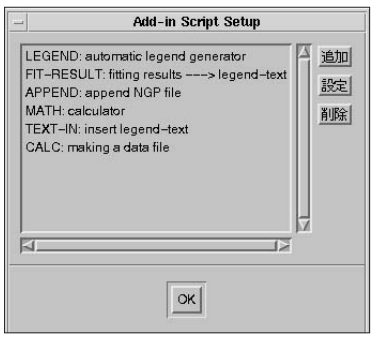


図9 アドイン・スクリプトを登録するためのダイアログ・ボックス

Ngraph [10] p89-96 99.08.19 17:38 ページ 93

波形データの作成

この解析では１列目に時間、２列目に波形データが入ったデータ・ファイルを取り扱う。実験ですでに用意されていればそれを使えばよいが、ここでは例となる波形データもCALCアドインを使って作成しよう。前述のように「グラフ」→「アドイン」で「CALC」を選び、**図7**のダイアログ・ボックスの「Output」にはexample.datを入力する。また「Formula」には、

```
SIN(2*PI*X)+COS(4*PI*X)+SIN(6*PI*X)+(RAND(2)-1)
```

を入力する。RAND（N）は０～Nの乱数を発生する関数で、ここでは疑似的にノイズの効果を表している。「Minimum」と「Maximum」はそれぞれ“０”と“５”を、「Division」には“256”を入力する（FFTで扱えるデータの個数は２の累乗である）。「Include max」のチェックは外して「OK」ボタンを押すと、**図10**のグラフが作成される。RAND（N）の関数を使っているので、描画するたびに波形が変わるはずである。ここでは、縦軸を回路の電圧、横軸を時間（t）と見たて、回路の電圧は1Hz、2Hz、3Hzの振動にノイズが加わったものである（もちろん実験でデータを得た場合には、振動成分はまだ分からない）。

前述のように、CALCアドインではexample.datに上式の計算結果は入っていないので、メニューから「出力」→「データファイル」でファイル名を指定し（fft.datとしよう）、解析に使うデータ・ファイルを作成する。example.datはもう必要ないので閉じておく。

振動解析

メニューから「グラフ」→「アドイン」でFFTアドインを起動すると**図11**のダイアログ・ボックスが表示される。「Input」に“fft.dat”を「Output」に“fft2.dat”を入力し、「OK」ボタンを押してNgraphに戻ると、fft2.datが３回描画ファイルとして開かれているはずである。fft2.datの１列目には周波数が、２列目には振動成分の実数部が、３列目には虚数部が入っている。グラフには実数部が青で、虚数部が赤で、大きさ（絶対値）が黒で表示されていると思う。それぞれを３つに分けたグラフを**図12**に示した。ここでは、スペクトル・アナライザーの出力らしくなるように、プロット・タイプは「bar_solid_fill_y」にした。棒の横幅はファイル・ダイアログ・ボックスの「サイズ」で変更できるので、隣と重なり合わないように適切に設定してほしい。FFTを行った後のデータの横軸は周波数（f）で、縦軸はexp（2πift）の振動成分を表している。絶対値のグラフから1Hz、2Hz、3Hzに大きなピークがあり、ノイズがのった波形から振動成分を分離で

図11　FFTアドインのダイアログ・ボックス

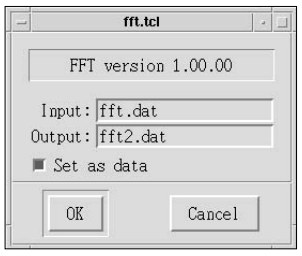


図10　振動波形のグラフ

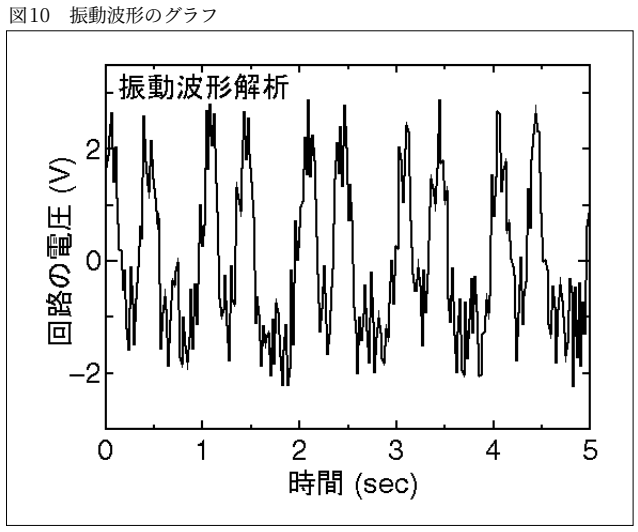
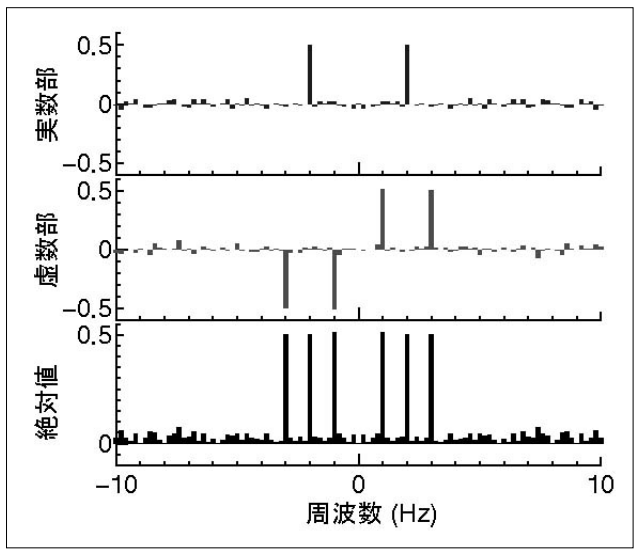


図12　図10の振動波形を高速フーリエ変換で解析したグラフ



きた。

Ngraphスクリプト

FFTアドインは、フィルタのfftを使ってデータ処理を行った。このフィルタを変更すれば、さまざまなデータ処理に応用できるので、アドインの作成方法を簡単に解説しよう。

アドインとNgraphのインタフェースや、グラフの保存形式にはNgraphスクリプトが使われている。これはshスクリプトのサブセットになっていて、shスクリプトの機能から、ジョブ制御とファイル・ディスクリプタ指定によるファイル入出力の機能だけを取り去ったものになっている。さらに、Ngraphの設定状態を取得したり、変更したりするための機能（コマンド）が追加されている。

Ngraphはオブジェクトの集合からなっている。たとえば、データ・ファイルを扱うのはfileオブジェクト、テキストの凡例を扱うのはtextオブジェクトといった具合だ。メニューから「グラフ」→「Ngraphシェル」を選ぶと、Ngraphスクリプトのインタープリタが起動するようになっている。そこで、

```
object
```

とコマンド入力すると、オブジェクトの一覧が表示されるので試してみてほしい。データ・ファイルを１つ開くと、fileオブジェクトの実体（インスタンス）が１つ生成される。各インスタンスには０からの通し番号が付けられる。これはFile Windowの左端に表示される番号だ。インスタンスの生成、削除は、それぞれnew、delコマンドである。データ・ファイルに関する設定は、各インスタンスのフィールド値の変更で行われている。設定は“＝”を使って、

```
オブジェクト名:インスタンス番号:フィールド名＝値
```

で行う。たとえば、ファイル名を保持しているのはfileオブジェクトのfileフィールドなので、

```
new file
file::file=example.dat
```

はデータ・ファイル（example.dat）を描画ファイルとして開くことに相当する。正確にいうと、２行目はfile:0:file= とインスタンスの番号を指定すべきなのだが、省略すると直前に操作（この場合new）されたインスタンスと解釈されるようになっている。グラフを保存したファイルもNgraphスクリプトなので、上記のような形式で記述されている。また、shの機能のforを使って、

```
for i in *.dat
do
  new file
  file::file=$i
done
```

と「グラフ」→「Ngraphシェル」で入力すれば、カレント・ディレクトリでdatの拡張子を持つファイルすべてを描画ファイルとして開ける。設定項目の参照は“$”を使って、

```
${オブジェクト名:インスタンス番号:フィールド名}
```

とする。たとえば、

```
echo ${file:0-!:file}
```

とすれば、開いたデータすべてのファイル名が表示される。ここではインスタンス番号の指定に範囲指定子“-”を使って、すべてのインスタンス（“！”は最後のインスタンス番号に置換される）を指定している。各オブジェクトのフィールドは、

```
object オブジェクト名
```

のように、objectコマンドの引数にオブジェクト名を指定すれば表示される。オブジェクトのメンバ関数はコマンドとして、

```
オブジェクト名:インスタンス番号:フィールド名 引数
```

で実行できる。たとえば、fileオブジェクトにはデータを取得するためのメンバー関数（後述）などが用意されている。

アドイン・スクリプト

アドインとNgraphのインタフェースもNgraphスクリプトで記述されている。**リスト１**はFFTアドインのfft.nscである。systemオブジェクトのtemp_fileフィールドを参照するとテンポラリ・ファイル名が取得できるようになっている。①で変数SCRIPTに取得したテンポラリ・ファイル名を代入している。②でwishを起動しfft.tclを実行する。コマンド・ラインにはフィルタのfftと、テンポラリ・ファイルを指定している。

fft.tclは第１引数から取得したフィルタ・コマンドfftを実行してフーリエ変換のデータ・ファイルを作成し、第２引数から取得したテンポラリ・ファイルに**リスト２**のような内容を書き込む。このテンポラリ・ファイルは③でNgraphスクリプトとして解釈・実行される（shellはNgraphスクリプトを扱うオブジェクトで、そのメンバ関数shellを実行している）。つまり、このテンポラリ・ファイルを介してfft.tclはNgraphの設定を変更できるのである。**リスト２**では出力ファイル（fft2.dat）を３回newで開いている。xとyのフィールドでプロットに使うデータ列を指定し、typeフィールドでプロット・タイプを指定している。R、G、Bフィールドはプロットする色で、math_yでY-軸のデータの変換数式を指定している（複素数の絶対値となるように変換する）。テンポラリ・ファイルの解釈・実行が終わると、fft.nsc（**リスト１**）は④で不要になったテンポラリ・ファイルを削除し、⑤でグラフを消去する（menuはNgraphのGUIを扱うオブジェクトである）。

データをアドイン・スクリプトで扱う

波形解析用のデータ（fft.dat）を作成する際、数式で変換された後のデータを得るために、CALCアドインでexample.datを作成し、さらにメニューから「出力」→「データファイル」を使った。アドイン・スクリプト中ではデータ・ファイルを直接扱えるので、この操作も自動化できる。**リスト３**はそのように変更したアドイン・スクリプト（fft.nsc）である。CALCアドインを実行した後、example.datが０番目のファイルとして開かれている。ここで**リスト３**のアドインを実行すると、fft2.datが作成され振動成分の絶対値のグラフが表示されるようになっている。

リスト２　fft.tclがテンポラリ・ファイルに書き出す内容。fft.tclが終了するとNgraphスクリプトとして解釈・実行され、Ngraphの設定が変更される

```
new file
file::x=1
file::y=2
file::file=fft2.dat
file::type=line
file::B=255
new file
file::file=fft2.dat
file::x=1
file::y=3
file::type=line
file::R=255
new file
file::file=fft2.dat
file::x=1
file::y=2
file::type=staircase_x
file::math_y='SQRT(%02^2+%03^2)'
```

リスト３の②でfileオブジェクトのopendataメンバ関数を実行し、データ・ファイルを入力ファイルとして開く。その後getdataメンバ関数を実行すると、data_x、data_yフィールドからデータを取得できる。このデータは変換数式で変換された後のデータである。そのほか、データ・ダイアログ・ボックスで指定できる、データ列の指定とかファイルの読み込み方法などの設定がすべて反映されている。これを③でテンポラリ・ファイルに書き出している。④でフィルタfftを直接（fft.tclを使わずに）実行しfft2.datを作成している。⑤で不要になったexample.datを閉じ、代わりに⑥でfft2.datを描画ファイルとして開き、プロット方法などを設定している。

また、dialogオブジェクトには、GUIで１行入力するためのテキスト・ボックスとか、ファイル・ダイアログ・ボックスを開いてファイル名を選択できるようなメンバ関数が用意されている。たとえば、**リスト３**の①を、

```
new dialog
OUTPUT=${dialog::input:'Output file'}
del dialog
```

に変更すれば、出力ファイルをダイアログ・ボックスで指定

リスト１　アドイン・スクリプトfft.nsc

```
SCRIPT=${system:0:temp_file}                          ——①
wish $NGRAPHLIB/fft.tcl $NGRAPHLIB/fft "$SCRIPT"      ——②
if [ -f "$SCRIPT" ];
then
  new shell                                           ——┐
  shell::shell "$SCRIPT"                              ——③
  del shell                                           ——┘
fi
system:0:unlink_temp_file                             ——④
menu:0:clear                                          ——⑤
```

リスト３　データを直接扱うアドインの例。GUIを使わないFFTアドイン

```
OUTPUT=fft2.dat                                       ——①
TEMPFILE=${system:0:temp_file}
file:0:opendata                                       ——②
while file:0:getdata
do                                                    ——┐
  echo ${file:0:data_x} ${file:0:data_y}>>$TEMPFILE   ——③
done                                                  ——┘
file:0:closedata
$NGRAPHLIB/fft $TEMPFILE $OUTPUT                      ——④
del file:0                                            ——⑤
new file                                              ——┐
file::file=$OUTPUT                                    ——⑥
file::x=1                                             ——⑥
file::y=2                                             ——⑥
file::type=staircase_x                                ——⑥
file::math_y='SQRT(%02^2+%03^2)'                      ——⑥
system:0:unlink_temp_file                             ——┘
menu:0:clear
```

できる。このようにNgraphスクリプトだけでもかなりのことができる。ちょっとしたアドインを作成するだけで、かなりのグラフ化作業が効率アップすると思う。

コマンド・ライン・オプションの追加

コマンド・ライン・オプションは/usr/local/lib/Ngraph/.Ngraphに渡され、そこで解析されている。このファイルもNgraphスクリプトで書かれていいるので、編集して独自のコマンド・ライン・オプションを追加できる。例としてグラフを保存したファイルを印刷するオプション(-p)を追加しよう。それには**リスト4**を/usr/local/lib/Ngraph/.Ngraphの36行目以降に追加する。使い方は、

```
% ngraph -p<グラフを保存したファイル>
```

である。また、コマンド・ラインだけの使用を考えて、印刷したあとはGUIを起動せずにNgraphが終了するようにしてある。

リスト4では、グラフの読み込みと描画を行っているが、簡単にその内容を解説しよう。①で初期状態のグラフを消去している。②でグラフが読み込むために、グラフを保存してあるファイルをスクリプトとして実行している。③はグラフ中のテキストの位置が正しく計算されるようにするために必要だ。④で印刷するためのオブジェクトであるgra2prnのインスタンスを生成し、ドライバ(gra2ps)と印刷コマンドを設定している。グラフの描画に関する制御を行っているのはgraオブジェクトであり、⑤で出力先をgra2prnに設定している。graオブジェクトのopen、draw、closeメンバ関数の実行で⑥グラフの描画が行われる。出力先がgra2prnなので、これで印刷されることになる。GUIが起動しないように、最後の⑦でNgraphを終了させている(systemオブジェクトのインスタンスを消去するとNgraphは終了するようになっている)。これらグラフの読み込みと描画はいろいろと応用範囲が広い。たとえば、アドイン・スクリプトの最後に同様の描画コマンドを記述しておけば、アドインの終了時に自動的に描画させるようもできる。

リスト4 ngpファイルからGRAファイルを生成するコマンドライン・オプション。.Ngraphの36行目に追加する

```
-p)
    for i in `derive -instance draw`          ┐
    do                                        │
      del ${i}:0-!                            ├ ①
    done                                      │
    if [ `exist gra:viewer` != 0 ]            │
    then                                      │
      del gra:viewer                          │
    fi                                        ┘
    . $2                                      ── ②
    new gra2null name=DUMMY                   ┐
    new gra device=gra2null:DUMMY             ├ ③
    gra::open                                 ┘
    new gra2prn name=OUTPUT                   ┐
    gra2prn::driver=gra2ps                    ├ ④
    gra2prn::prn="| lpr"                      ┘
    gra:viewer:device=gra2prn:OUTPUT          ── ⑤
    gra:viewer:open                           ┐
    gra:viewer:draw                           ├ ⑥
    gra:viewer:close                          ┘
    del system:0                              ── ⑦
;;
```

最後に

今回はアドインの使い方と作成方法を中心に解説をした。また、伊東宏之氏によるWebページ「Ngraph script活用法」、

```
http://member.nifty.ne.jp/slokar/ngraph/
```

にもスクリプトの解説や、便利なアドインが紹介されているので、ぜひとも参照してほしい。

以上、Ngraphの機能を一通り早足で解説してきた。Ngraphは2次元散布図だけを作成するソフトである。たかが2次元の散布図なのではあるが、それは数値データを視覚的に表現したものであり、表現する者(研究者や技術者)によってさまざまな形態が存在する。それらの多くは、前回説明したテクニックを組み合わせれば作成可能だと思うが、まだほかにも新たな機能が必要かもしれない。そのような要望や、使ってみた感想などを、

```
isizaka@msa.biglobe.ne.jp
```

あてにぜひとも送ってほしい。

最後にNgraphの開発に当たって協力してくださった方々に感謝の意を表したい。