ワークステーション入門—X-Windowでお絵書きを

大塚, 寛
九州大学理学部数学教室数理解析学講座

https://doi.org/10.15017/6769102

出版情報：情報処理教育広報. 15 (2), pp.3-16, 1992-12. Educational Center For Information Processing, Kyushu University
バージョン：
権利関係：

ワークステーション入門 – X-Window でお絵書きを

理学部 数学教室 数理解析学講座
大塚 寛

1 X-Window

ワークステーションを立ち上げた、あるいは他人が使った後のディスプレイはどうなっているでしょうか。白地に黒の文字で、文字が大きいとか画面のサイズが大きいとかの違いはありますが、基本的にパソコンのディスプレイと同じ状態で、コンソール画面と呼ばれます。もちろんこの状態で Unix を使うことも出来ますが、それはパソコンから Unix にログインして使うのと同じ事です。

しかし、ワークステーション上には便利なウィンドウ環境が存在します。図1は筆者がワークステーションにログインした直後のディスプレイですが、時計 (xclock) や電卓 (xcalc)、パソコンのひとつの画面にあたるターミナルエミュレータ (kterm) 等が表示されています。ウィンドウ環境では、このようにディスプレイ上に複数のプロセスを同時に表示したり、文字以外の情報も表示出来ます。

ワークステーションには、X-Window, SunView, OpenWindows といった3種類の異なるウィンドウ環境が提供されていますが、ここでは最も汎用性が高いと思われる X-Window を用います。

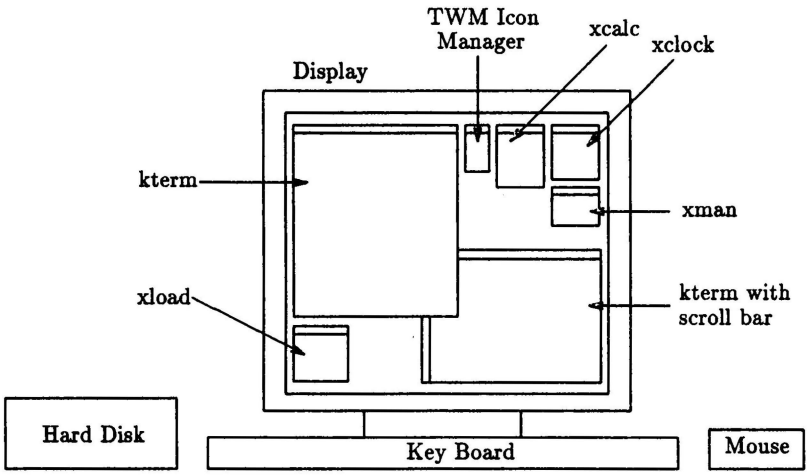


図 1: ワークステーションとこれをウィンドウ環境で使用する例

なお、図1は皆さんが最初に立ち上げた時のディスプレイとは異なるはずですが、このように自分の趣味に合ったディスプレイに簡単に設定できるので、いろいろ試して気にいったディスプレイにして下さい。

2 X-window 上の図形編集プログラム

コンソール画面では表示の対象となるのは文字だけです。ここでは特殊なことをしない限り、テキストやプログラムを書くなど基本的に文字の編集しかできません。しかし、ウィンドウ環境では図形も表示の対象となります。そこで、このウィンドウ環境で文字だけでなく図形の編集も出来れば、ウィンドウ環境の利用価値が広がります。

X-Window 上には、**xfig, xkey3, tgif+** 等のウィンドウ環境を利用したドロー系の図形編集プログラムがあります。また主にグラフを対話的に描画することを目的とした **gnuplot** というプログラムもあります。

もちろん、これらのプログラムで描かれた図形は紙に出力できてこそ利用価値があるわけです。これらのプログラムは、編集した結果を直接あるいは間接的に PostScript 形式でファイルに出力出来るようになっているので、直接プリンタに出力、あるいは **X-Window** 上で **gs** というプログラムを使ってプレビュー出来ます。

しかしここでは、その他にも文書処理システムである LaTeX の **picture** 環境で利用出来る等、利用範囲が広いと思われる **xfig** および **gnuplot** の使い方を、LaTeX へのデータの変換も交えながら説明します。

なお、**xfig** はディレクトリ `/usr/local/bin/X11` 以下に **X-Window** 関係のコマンドと一緒にあり、**gnuplot** は `/usr/local/bin` 以下にあるので、これらのディレクトリへの path は設定されているものとします。

3 xfig

教育センターで使える **xfig** は Version 2.0 Protocol Version 2.0 Patch Level 9 です。最新の **xfig** ではありませんが十分使えます。

xfig を立ち上げるには、シェルのコマンドラインから `xfig` と入力するか、ウィンドウマネージャーである **twm** のメニューに組み込み、ここから選んで立ち上げます。しかし **xfig** には立ち上げ時のオプションがありますから、いろいろオプションを確かめて、気に入ったオプションが決まってからメニューに組み込むのがいいでしょう。以下のオプションが便利と思われます。

`-ri` 描画や編集用のコマンド領域をキャンバス領域の右に表示。(デフォルトは `-le` で左に表示)

`-me` 図形の単位をセンチ (cm) にして利用する。(デフォルトは `-inc` で単位はインチ)

`-inv` 白黒を反転して表示する。

図 2に **xfig** で図 1を編集している簡単な画面を載せているので、以下の説明の参考にして下さい。

3.1 xfig の画面

xfig を立ち上げると、

> **FIG : FACILITY FOR INTERACTIVE GENERATION OF FIGURES V.2.0 Protocol V.2.0 Patch Lev. 9**

というメッセージがディスプレイの左上に現れ、しばらくして **xfig** のウィンドウが現れます。オプションなしで立ち上げた場合、このウィンドウには、左側に描画や編集用コマンドの領域が、下部にメッセージの表示領域、文字入力の設定領域及びとファイル操作関係のコマンドの領域があり、上部と右側には inch 単位 のスケールが現れます。(図 2参照)

各コマンドは、それを示すアイコン上でマウスの左ボタンをクリックすることで選択されます。同時にアイコンが反転表示され、メッセージ表示領域にコマンド名が表示されます。ここはまた、コマンドがファイル名やディレクトリ名等を必要とする時に、それらを入力する領域にもなります。

中央のキャンバス領域では、これらのコマンドを使って図形の描画や編集を、基本的にマウス操作だけで行ないます。ここではマウスの右ボタンがポップアップメニューに割り当てられています。

3.2 描画機能

描画には、**円、楕円、スプライン曲線、閉スプライン曲線、線分、多角形、長方形、角が丸い長方形、文字入力、円弧**があります。

アイコンはそれらが表す図形になっており、どういったコマンドかすぐに分かるのでこれらの説明は必要ないでしょう。“習うより慣れろ”で思考錯誤して憶えるのが早道です。

これらの操作には一般的な規則があり、理解しやすいようになっています。始点、(必要ならば) 中間点はマウスの左ボタンのクリックに、終点は中央ボタンのクリックに割り当てられています。文字入力だけは左ボタンクリックの後、キャンバス領域上で文字を入力してからリターンキーあるいは中央ボタンのクリックで一行の入力となります。

3.3 編集機能

図形の編集には、**直線とスプライン曲線の相互変換、図形領域の設定 / 解除 / 拡大 / 縮小、矢印の設定 / 解除、中間点の追加 / 削除 / 位置変更、図形及び図形領域の移動 / 複写 / 削除、垂直水平線を基準とした図形及び図形領域の鏡面対称 (移動 / 複写)、図形の回転 (移動 / 複写)、図形のパラメータ設定**があります。

図形の編集で便利な機能は、**図形領域の設定 / 解除**でしょう。図形領域は長方形の描画と同じ要領で設定できます。ただし、このことは図形領域が長方形に制限されることも意味しているので注意して下さい。この領域内の複数の図形はまとめて操作することが出来るので、複雑な図形を複写 / 移動したり、回転させたりするには便利です。これがないと大変なのは、図形領域を設定せずに、複雑な図形を拡大 / 縮小させてみれば実感できるはずです。

ここでのほとんどの操作も左ボタンだけ、あるいは左ボタンで始めて中央ボタンで終る様になっていますが、矢印の設定 / 解除は左ボタンで設定 / 右ボタンで解除、図形の鏡面対称、回転は左ボタンが複写 / 中央ボタンが移動になっているので注意して下さい。マウスのボタンの説明はメッセージ表示領域にも表示されます。

3.4 表示設定機能

図形の描画は表示設定に示される設定値に従って行なわれます。ここでは、**角度の制限、線種の選択、方眼の設定 / 解除、始点 / 終点での矢印の設定 / 解除、図形の各点の位置の制限、領域塗りつぶしとその塗りつぶしパターンの設定、線幅の設定、角の丸さの設定**が出来ます。

角度の制限には、設定なし、**LaTeX 線分対応、LaTeX 矢印線対応、水平垂直と斜め 45 度のみ、水平垂直のみ、斜め 45 度のみ**があります。図形を LaTeX に取り込んで使いたい場合は LaTeX 線分対応か LaTeX 矢印線対応にしておくと便利です。

方眼はオプションで指定した単位で表示されます。正確に図形を書きたい場合や、位置がずれているとか大きさが違うと思ったら、一度この設定を利用して下さい。また、各点の位置の制限もこういった時に役に立ちます。

3.5 文字設定機能

文字の入力は、フォント、文字サイズ、テキストの行間、位置揃えの指定に応じて行なわれます。

フォントは **X-Window** 上で利用されるフォントを使います。フォントにはいろいろな種類がいろいろなサイズで用意されていますが、残念ながら **xfig** では日本語は入力できないので、直接プリンタに出力する場合の文字には日本語を含めることは出来ません。

しかし LaTeX に取り込んでしまえば日本語も使えるので、**xfig** が出力したファイルを LaTeX に変換した後で日本語に書き直せばいいでしょう。図 2はそのようにしています。ただし、今度は LaTeX のフォントが **X-Window** 程には用意されていないので、**xfig** で使えたフォント (及び文字サイズ) がすべて使えるとは限りません。あるいは **xfig** の内容に日本語を含めるだけなら、以下に説明する方法もありますが、ある程度 PostScript 言語や日本語 PostScript のフォントの知識が必要になります。

3.6 その他の機能

その他にも、**終了、ファイルの読みだし、書き込み、編集、プリンタの設定、出力、用紙のポートレイト / ランドスケープモードの設定**そして**アンドゥ、再描画機能**があります。

アンドゥは、キャンバス領域を直前の操作の前の状態に戻します。失敗したと思ったら、慌てずアンドゥして下さい。ただし、何度も前の操作の間違いを遡ってアンドゥで取り消すことは出来ません。アンドゥのアンドゥは何もしないのと同じです。また、再描画は図形の削除等で画面上の隣接した図形の一部の表示が消えたり等、画面が乱れた時に実際の内容を画面に表示し直してくれます。再描画のアンドゥはその直前の乱れた画面ではないので安心して下さい。

xfig は標準で PostScript プリンタへ出力しますが、プリンタの設定をファイルに変更することも出来ます。**xfig** で描いた絵だけで十分だが、フォントの制限を受けずに日本語を使いたいという人は、ここで出力された PostScript 形式のファイルの内容に日本語を含めるなどの変更を行ない、プリンタに出力するのもいいでしょう。PostScript 形式のファイルは通常のテキストファイルですから、テキストエディタで編集出来ます。このとき **gs** でプレビューを行ない、思い通りの絵が出来ているか確かめてからプリンタに出力する方が紙の無駄使いにもならずにすみ、地球にやさしい使い方が出来ます。なお、一度 PostScript 形式で出力されたファイルは **xfig** では読み込めません。

複雑な図形を描く時は時間もかかり、1日で終るとは限りません。また、描き終ってプリンタに出力し終っても描いた図形が必要なくなるとは限りません。そういう時は描いた図形をファイルに保存し、次に **xfig** を使う時にそのファイルを編集します。更に、ひとつの画面に複数のファイルを読み込むことによって、いくつかの別々に描いた図形をひとつにまとめ上げることが出来ます。

またこれは普通のテキストやプログラムを書く時もいえることですが、複雑な図形を描く時はなるべくこまめに、あるいは一区切りついた時点でファイルの保存を行ないましょう。アンドゥは1回しか出来ないし、いつ何時不慮の事故で **xfig** が突然使えなくなるなどの事態が生じるか分かりません。そういった事態への対応としては、こまめにファイルに保存することが基本中の基本です。

終了や書き込みでは、現在編集中の図形をファイルに書き出さずに終了するかとか、存在するファイルに上書きしていいかどうか聞いて来る時があります。それで良ければ左ボタンを、中止したければ中央か右ボタンをクリックするようになっています。

3.7 xfig から LaTeX へ

この手の図形編集ソフトの解説を文章でぐだぐだ書いたものを読んでも、なんのことやら分からないと思います。やはり、実際に **xfig** を使っていろいろ絵を描いてみるのが一番です。そこでここでは、**xfig** で描いた図形を LaTeX に取り込む方法を紹介します。

ただしまず注意しなければならないのは、**xfig** で描いた図形と全く同じ図形が LaTeX で得られるわけではないことです。ある程度の制限がつくことを憶えておいて下さい。

xfig が出力したファイルは FIG コードと呼ばれるフォーマットの通常のテキストファイルになっています。このフォーマットのファイルを直接 LaTeX に取り込んでも図形にはなりませんが、**fig2dev** というプログラムを使って LaTeX の **picture** 環境のコマンド、あるいはこれを更に拡張した **epic** のコマンドに変換すると、LaTeX で図形として出力出来ます。

fig2dev の書式は以下の通りです。[ ] で括られたオプションあるいはファイル名は省略可能です。

`fig2dev -L` *language* [`-m` *mag*] [`-f` *font*] [`-s` *fsize*] [*other options*] [*fig-file* [*out-file*]]

fig2dev は *fig-file* で指定したファイルに書かれた FIG コードを *language* で指定されたグラフィック用のコマンドに変換し、その結果を *out-file* に書き出します。*fig-file*/*out-file* は省略されれば、各々標準入力 / 標準出力になります。

各オプションは次の意味を持ちます。

`-L` *language* 出力するグラフィック用のコマンド (*language*) を指定します。*language* で指定できるコマンドは、box, epic, eepic, eepicemu, latex, null, pic, pictex 及び ps のいずれかで、必ず指定しなければなりません。

`-m` *mag* 拡大率を *mag* に設定します。デフォルトは 1.0 です。

`-f` *font* テキストのデフォルトのフォントを *font* に設定します。デフォルトはローマン体です。このオプションは各 *language* に依存します。例えば epic や latex 等の LaTeX ベースの *language* の場合、ファイル `lfonts.tex` に記述されたフォントの base を *font* に指定します。

`-s` *fsize* テキストのデフォルトの文字サイズを *fsize* に設定します。デフォルトは 11 × *mag* で、`-m` オプションで拡大、縮小されます。

other options このオプションは各 *language* に依存します。ここでは epic と latex の中で役に立ちそうなオプションを取り上げます。

- `-l` *lwidth* epic の場合、線幅が *lwidth* より大きければ `\thicklines` を使う。デフォルトは 2。
  latex の場合、細い線と太い線の境界を *lwidth* ピクセルに設定する。LaTeX は 2 つの線幅 `\thicklines` と `\thinlines` しかサポートしない。線幅が *lwidth* より大きければ `\thicklines` を使う。これは、点線のドットのサイズにも影響する。デフォルトは 0。
- `-W` 可変な線幅を許す。(epic のみ)
- `-w` 可変な線幅を許さない。(epic のみ)
  このときは、`\thicklines` と `\thinlines` コマンドのみが出力として生成される。

LaTeX はたとえ epic を使っても xfig が生成するすべての図形を表現出来るわけではありません。例えばエリアフィルは出来ないし、スプライン曲線は出来る限り直線で近似しようとするだけです。xfig の方で LaTeX 用の制限を設定しておく、あるいは制限を意識して図形を描いた方が良いでしょう。

以上が fig2dev のオプションです。例えば sample.fig というファイルを、コマンドは epic で、フォントは cmb を、文字サイズは 10 で出力したければ、

```
% fig2dev -L epic -f cmb -s 10 sample.fig > sample.tex
```

とすればいいわけです。後はこの sample.tex を適当に編集し、以下のように別の LaTeX ファイルから \input コマンドで読み込めば、LaTeX で書いた文章中に xfig で編集した図形を取り込めます。なお以下の LaTeX の例では % 以降は改行まで注釈文です。

```
\documentstyle[...,epic,...]{jarticle}    % epic スタイルファイルの指定が必要
    ⋮
\begin{document}
    ⋮
\begin{figure}[htbp]                      % この位置に出力
    \begin{center}                        % センタリング
        \input{sample}                    % ここで読み込む
    \end{center}
    \caption{sample.fig}                  % キャプションもつけておく
\end{figure}
    ⋮
\end{document}
```

但し図形の編集は LaTeX だけで行なうのは大変なので、図形は xfig で完成させておいて、LaTeX の方での編集は、図の大きさ、文章中の位置、図の中のテキストの変更程度にとどめておいた方がいいでしょう。

なお、fig2dev には transfig という Makefile を生成するフロントエンドのコマンドがあります。これを使うと以後は make とタイプするだけで fig2dev が自動的に実行されます。また、Makefile を編集して jlatex 等の起動も含めることにより、更に自動化できます。

3.8 xfig のバグレポート

xfig にはいくつかのバグが存在します。ここではマニュアルに記載されている xfig のバグを挙げます。もちろん、以下に挙げるバグ以外にもまだバグが潜んでいるかも知れませんが、よほど特殊な使い方をしない限りそのようなバグに出会わすこともないでしょう。

1. テキスト入力はアンドゥの対象にならない。
2. 閉スプライン曲線のエリアフィルは、キャンバス領域上では見えないが、プリントした場合はちゃんと現れる。
3. 図形領域の設定のアンドゥが出来ない。
4. 長方形の点移動のアンドゥは元の長方形に戻らない場合がある。
5. 狭過ぎる楕円は正しく描けない。

以上のことに注意すれば、**xfig** は十分使えるプログラムです。またここでは紹介しなかった **tgif+** や **xkey3** も **xfig** と同様の機能を持った図形編集プログラムです。いろいろ使ってみて自分の好みにあったプログラムを選んで下さい。

またここでは、ほとんど LaTeX の **picture** 環境との関係を中心に解説しましたが、もちろん単独で本来の制約のつかない使い方もできます。例えば OHP シートに図を書くぐらいで、特に LaTeX が必要であるわけではない方にも十分利用価値があるので、図形を描いてみたいと思ったらこれらのプログラムを利用して下さい。

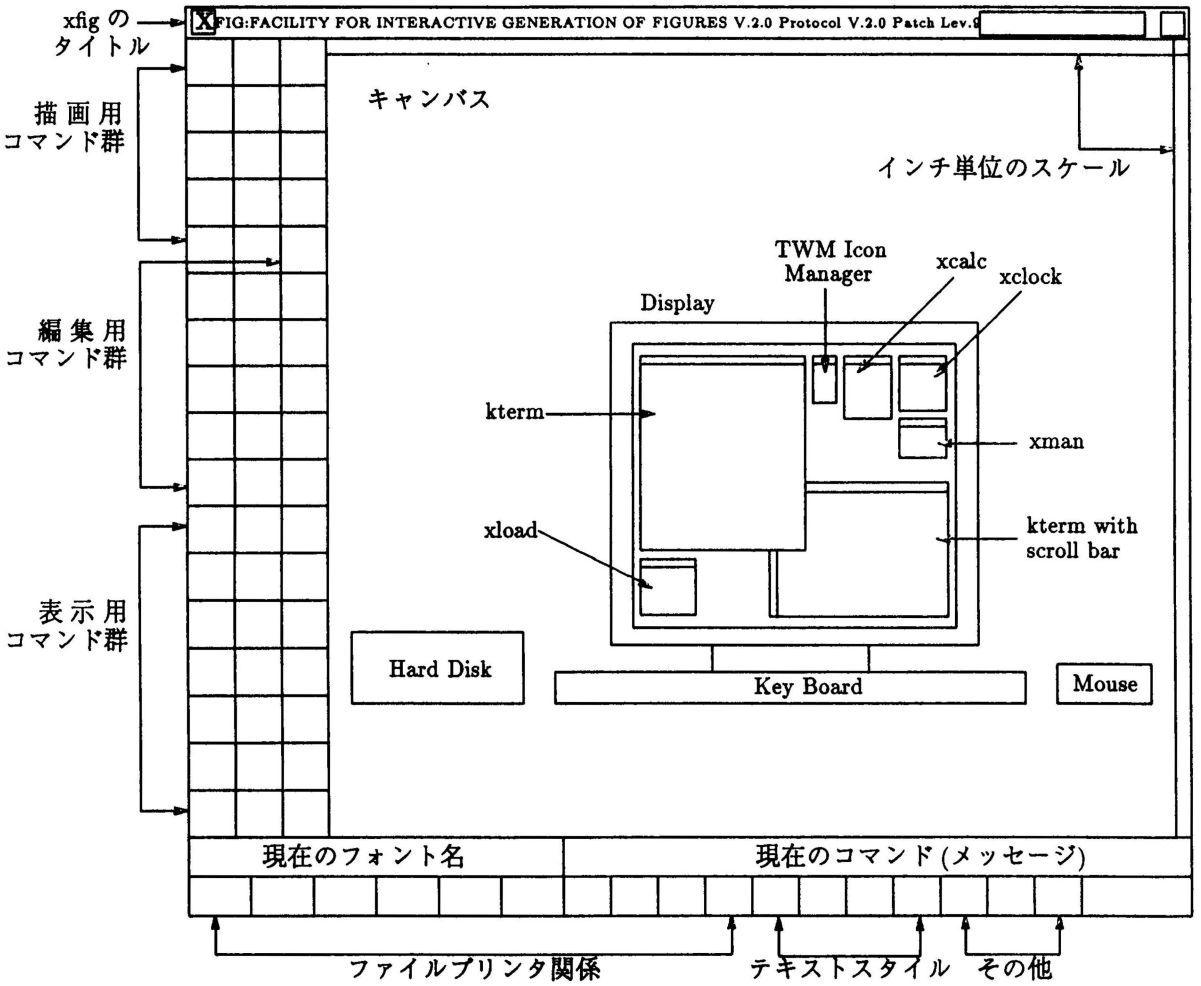


図 2: **xfig** で図形を編集する例

4 gnuplot

gnuplot はシェルのコマンドラインから gnuplot と入力することで起動します。**gnuplot** は先に解説した **xfig** あるいは **xkey3**, **tgif+** とは異なり、**gnuplot** のコマンドをキーボード / ファイルから入力しながらグラフをプロットするようになっているので、メニューから起動する場合も **kterm** 等のターミナルエミュレータを介しての起動となります。

出力は **X-Window** だけではなく、通常のディスプレイ、あるいはファイルもサポートして

おり、その出力形式も X-Window や通常のディスプレイでの表示そのものの他、PostScript や LaTeX の picture 環境もサポートしています。従って、最終結果を紙に出力する場合は、まず X-Window 上でグラフをプロットし、その結果を確かめてから、プリンタやファイルへ PostScript や LaTeX の picture 環境で出力すればいいでしょう。

また gnuplot にはオンラインの日本語ヘルプ機能があります。コマンド等の使い方が分からなくなったら積極的に利用しましょう。もちろん、この時は当然日本語が扱えるターミナルエミュレータ (kterm 等) でなければなりません。

なお、各コマンドの説明はこのヘルプ機能を使って調べることが出来るので、ここではヘルプ機能以外は必要最小限の説明にとどめます。

4.1 gnuplot の画面と行入力

gnuplot を立ち上げると次のようなメッセージが現れ、プロンプト gnuplot> が現れてコマンド入力待ちになります。gnuplot を X-Window で使う場合、グラフのプロットには別のウィンドウが用意され、入力されたコマンドにしたがって、そこに表示されます。

```
        G N U P L O T
        unix version 3.2
        patchlevel 2, Mar 24 92
        last modified Sat Mar 24 21:08:47 PST 1991

Copyright (C) 1986, 1987, 1990, 1991, 1992  Colin Kelley, Thomas Williams

        Send bugs and comments to ken@ec.kyushu-u.ac.jp

Terminal type set to 'x11'
gnuplot>
```

gnuplot はシェルと同じようにコマンドを入力しなければなりませんが、以下のような簡単なコマンドライン編集機能およびヒストリー機能を持っています。これによって現在の入力行を編集したり、過去のコマンドを呼びだして編集し、再実行することができます。入力を間違えたとか、以前と同じような入力を繰り返す時などに利用して下さい。

^B	一文字左へ移動。
^F	一文字右へ移動。
^A	行頭へ移動。
^E	行末へ移動。
^H,DEL	カーソルの左の文字を削除。
^D	カーソル位置の文字を削除。
^K	カーソル位置以降の文字をすべて削除。
^L,^R	画面が乱れた場合などに行を再表示。
^U	行全体を削除。
^W	カーソルの左の一語を削除。
^P	ヒストリーをさかのぼって検索。
^N	ヒストリーをくだって検索。

なお、^B はコントロールキーを押しながら B キーを押すことを意味します。コマンドラインの編集が終った後にキャリッジリターンキーを押すと、そのときのカーソルの位置にかかわり無く行全体がコマンドとして処理されます。

また gnuplot の終了は quit と exit です。

4.2 gnuplot のヘルプコマンド

gnuplot には多くのコマンドが存在し、それらをすべて解説するのは大変なので、ここではヘルプ機能の使い方を説明するにとどめます。各コマンドの説明はこのヘルプ機能を使って調べて下さい。ヘルプ機能は階層構造を持っているので、コマンド名が分からなくても、以下のように help から始めて求めるコマンドに到達できます。

プロンプト gnuplot> に続いて help と入力すると、以下のような内容が表示され、

```
GNUPLOT は、コマンド入力方式の対話的な関数描画プログラムです。コマン
ドや関数名は大文字小文字を区別します。いずれのコマンドも、あいまいさ
の無い限りにおいて省略することができます。一行中にはセミコロン (;) で区
切って複数個のコマンドを書くことができます。文字列は引用符を使って表
します。引用符は、一重のものでも、二重のものでも構いません。例 :

        load "filename"
        cd 'dir'
          ⋮
```

最後に、次にどのコマンドのヘルプに移れるか表示されます。

```
Help topics available:
        autoscale       bugs    cd      clear
        comments        environment     exit    expressions
        help    line-editing    load    pause
        plot    print   pwd     quit
        replot  save    set     shell
        show    splot   startup substitution
        userdefined
```

以上の内容の表示には、環境変数 PAGER で指定されたコマンドが使われます。

これを終了すると、Help topic: と聞いてきます。ここでキャリッジリターンだけを入力すれば、ヘルプ機能は終了し、gnuplot のプロンプト gnuplot> が現れ、コマンド入力待ちになります。また、? を入力すれば、Help topics available: に示してあるコマンド表示され、再び Help topic: を聞いてきます。

一方、Help topics available: に示してあるコマンドを入力すれば、そのコマンドのヘルプの内容が表示されます。この場合、更に別コマンドのヘルプが用意されていれば、同様の操作でそのコマンドのヘルプを表示できます。用意されていなければ、そのコマンドのヘルプが終了し、もとのヘルプでの Help topic: が表示されます。ここでまた別コマンドのヘルプに移動することも出来るし、終了させることも出来ます。

以上のようにヘルプ機能は階層構造を持っていますが、現在どのコマンドのヘルプかは、Help topic: が Subtopic of plot: のように変わるので、これで確認して下さい。

また、ヘルプ機能は help help のようにコマンドを指定することも出来ます。最初はヘルプのメッセージにあるように、help plot としてプロット用のコマンドを調べてみて下さい。

4.3 gnuplot の expression

gnuplot では関数のプロットのために、C や FORTRAN において利用可能な基本的な数

学表現を使用できます。複素数の定数は {< 実部 >,< 虚部 >} と表せます。ここで < 実部 > と < 虚部 > は整数や実数の数定数でなくてはなりません。

関数は、引数として特に断らない限り、整数、実数、複素数をとれることを除いては、Unix の数学ライブラリの対応する関数と同じものです。以下の関数が利用可能です。

abs	acos	arg	asin	atan	besj0	besj1	besy0	besy1	ceil	cos	cosh	exp
floor	gamma	imag	int	log	log10	real	sgn	sin	sinh	sqrt	tan	tanh

演算子は対応する C の演算子と同じもので、以下の演算子が利用可能です。引数としては、特に断わらない限り、整数、実数、複素数のいずれも利用可能です。括弧内は使用例です。

2 項演算子	**	(a**b)	*	(a*b)	/	(a/b)	%	(a%b)	+	(a+b)
	-	(a-b)	==	(a==b)	!=	(a!=b)	&	(a&b)	^	(a^b)
	|	(a|b)	&&	(a&&b)	||	a||b				
3 項演算子	? :	(a?b:c)								
単項演算子	-	(-a)	~	(~a)	!	(!a)	!	(a!)		

個々の関数及び演算子の意味についてはヘルプ機能を使って調べて下さい。

4.4 gnuplot から LaTeX へ

gnuplot はいろいろな出力形式でグラフをプロット出来ます。X-Window 上では terminal が x11 に指定してありますが、コマンド set terminal で出力形式の一覧、以下のように設定することで出力形式を変更できます。なお、出力先は標準で標準出力なので、出力をファイルに変更するときには set output を使用してください。

特に terminal として latex を、output として適当なファイル名を指定することで LaTeX の picture 環境でデータを出力することが出来ます。以下の例では、アンダーラインがついた部分が入力コマンドで、# 以降から改行までは gnuplot の注釈文です。

```
gnuplot> set terminal latex      # 出力形式を LaTeX に変更
Terminal typeset to 'latex'
gnuplot> set nokey               # なくても構わない
gnuplot> set output 'sin.tex'    # 出力ファイルは sin.tex
gnuplot> plot [-pi:pi] sin(x)    # [-π,π] の範囲で sin(x) をプロット
```

後は別の LaTeX ファイルから \input コマンドで読み込むことで LaTeX で書いた文章中に gnuplot でプロットしたグラフを取り込めます。

```
\documentstyle[...,epic,...]{jarticle}    % epic スタイルファイルの指定が必要
    ⋮
\begin{document}
    ⋮
\begin{figure}[htbp]                      % この位置に出力
    \begin{center}                        % センタリング
        \input{sin}                       % ここで読み込む
    \end{center}
    \caption{$y = \sin(x)$}               % キャプションもつけておく
\end{figure}
    ⋮
\end{document}
```

図 3はその結果です。また terminal として fig を指定すれば、xfig での利用も可能になります。

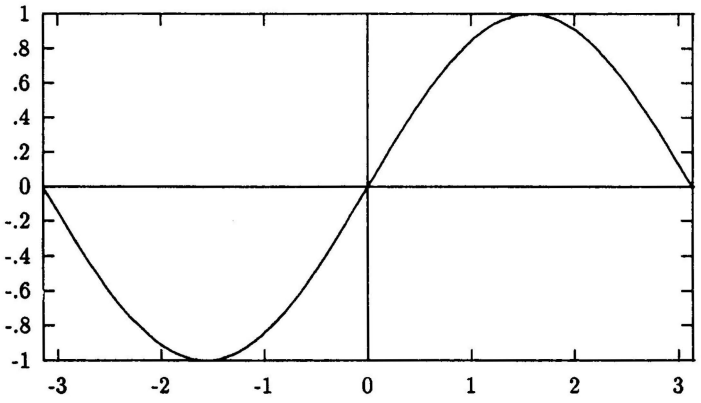


図 3: $y = \sin(x)$

4.5 外部データの取り込み

gnuplot では用意された関数の他にも、簡単な関数ならユーザーが **gnuplot** のなかで定義できるようになっています。しかし複雑な関数等を記述できるわけではないので、どうしても外部で作られたデータを利用するする機能がなくてはなりません。

この機能は `plot` と `splot` にあり、`help plot datafile` 等で参照できます。基本的にデータファイルには、1 行にひとつずつデータが書かれていなければなりません。`#` で始まる行はコメントとして扱われ、無視されます。`plot` に対しては、各データ点は (x,y) の組を、`splot` では (x,y,z) を表します。いずれの場合もデータファイル中の数字は、空白文字で区切られていなければなりません。この空白文字によって、各行は列に区切られます。ここでは例として、シェルソートの実行過程の表示を **gnuplot** で表示させてみます。

図 4は 64 個の 0 から 99 までのランダムな整数値をシェルソートする C のプログラムです。関数 **gpout()** が **gnuplot** のデータファイルの形式でデータを出力します。ここでは生成される順に `init.log`, `gap32.log`, `gap16.log`, `gap8.log`, `gap4.log`, `gap2.log`, `gap1.log` がデータファイルとなります。**gnuplot** での表示には、次のように入力するだけです。

```
gnuplot> set data style points                  # 各データの表示形式
gnuplot> plot 'init.log' title 'initail data'   # ファイル init.log のデータをプロット
gnuplot> plot 'gap32.log' title 'gap = 32'      # ファイル gap32.log のデータをプロット
gnuplot> plot 'gap16.log' title 'gap = 16'      # ヒストリーとコマンドライン編集機能を使おう
           ⋮
```

図 5はグラフのサイズの縮小を行なっていますが、基本的に同じようなグラフが表示されます。

4.6 バグレポート

gnuplot のバグは `help bugs` で参照できますが、ベッセル関数 /Γ 関数が複素数引数に対応していないことぐらいです。他にも stdio ライブラリのバグが載せてありますが、OS のバージョンが Release 4.1.1-JLE1.1.1RevB なので関係ないと思われます。

4.7 おまけ

いくら gnuplot が便利な機能を備えたグラフプロットプログラムといっても、どのようなコマンドがあって、それをどのように使ったらいいのか、最初から分かるはずはありません。またお遊び程度の簡単なグラフと違って、何もない状態から複雑なグラフを描こうとすると、結構思考錯誤しなければならず、それだけで疲れ果ててしまいます。

こんな時には gnuplot のデモを利用しましょう。gnuplot にはいろいろなグラフをプロットするデモのデータが付属しています。まずこれらのデモを gnuplot で実行してみて、気にいったデータや、役に立ちそうなグラフがあれば、それらを自分のディレクトリにコピーして内容を調べ、コマンドのパラメータを変えたり等、編集し直して gnuplot で実行して下さい。これでかなりの労力が節約出来、gnuplot を使いこなせるようになるでしょう。

デモのデータはディレクトリ /usr/local/src/gnuplot/demo 以下にあります。

```
gnuplot> cd '/usr/local/src/gnuplot/demo'    # /usr/local/src/gnuplot/demo に移動
gnuplot> load 'all.demo'                     # all.demo を実行 (他のデモを load するだけ)
```

またディレクトリ /usr/local/src/gnuplot/docs/latextut には LaTeX と合わせて使う方法を解説した LaTeX のソースファイルがあります。gnuplot を使ってみようと思われた方は、これらのデモや解説を大いに利用して下さい。

ところで、gnuplot はワークステーションでしか動作しないわけではありません。情報処理教育センターのパソコン FMR でも gnuplot が使えます。小講義室でワークステーションが利用できない場合は、FMR から telnet でワークステーションにログインして gnuplot を使うことも出来ますが、この時は terminal が dumb に設定され、キャラクタでしかグラフを表示出来ません。FMR のディスプレイにちゃんと表示したければ、この gnuplot を利用するのもいいでしょう。

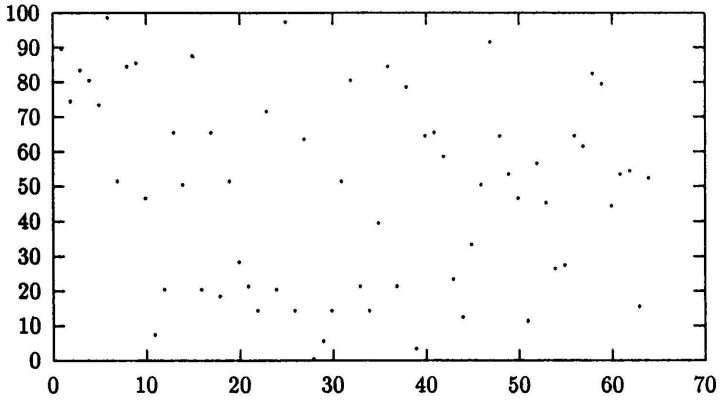
5 終りに

ここまで書いてきてふと読み返してみると、"X-Window でお絵書きを" といいながら、ペイント系のツールではなくドロー系のツールの解説で、それもほとんど LaTeX の picture 環境を強化するといった片よった話が中心になってしまいました。

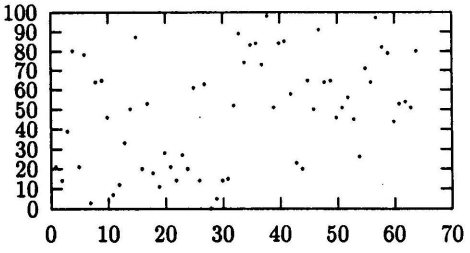
しかしレポートや資料を書く等、ワークステーションで文書処理をやろうと思ったら、どうしても LaTeX を使わなければなりません。またプログラムの実行結果が単なる数字の羅列よりは、グラフで図示した方が理解しやすいのも当然です。こういった時、ここで解説した xfig や gnuplot を利用したり、更に LaTeX に取り込んで文章と一緒にまとめあげるのが、教官の資料作成や学生の演習という観点からは、ワークステーションの一番身近かな使い方だと思います。この解説を読んで、一人でも多くの方がワークステーションを今までのホスト計算機やパソコンとは違った形で利用するようになってもらえれば幸いです。

```
#include    <stdio.h>
#include    <stdlib.h>
#define     SIZE    64
#define     MAX     100
int a[SIZE+1];                                  /* use a[1] - a[SIZE] like FORTRAN */
char *init = "init.log", *shell = "gap%d.log"; /* output filename */

main() {
  int i;

  srand(1);                 /* make inital date */
  for(i = 1; i <= SIZE; i++)    a[i] = rand() % MAX;
  gpout(init, a, SIZE);     /* output data for GNUPLOT */
  shellsort(a, SIZE);       /* shell sort */
}

gpout(fname, array, size)
char    *fname;
int     array[], size;
{
  int i;
  FILE    *fp;

  fp = fopen(fname, "w");
  fprintf(fp, "# GNUPLOT data file (shell sort)\n");
  for(i = 1; i <= size; i++)    fprintf(fp, "%3d %3d\n", i, array[i]);
  fclose(fp);
}

static char file[16];
shellsort(array, size)
int array[], size;
{
  int i, temp, gap, j1, j2;

  for(gap = size>>1; gap > 0; gap >>= 1) {
    for(i = gap+1; i <= size; i++)
      for(j1 = i-gap; j1 > 0; j1 -= gap) {
        j2 = j1 + gap;
        if(array[j1] <= array[j2]) break;
        else { temp = array[j1]; array[j1] = array[j2]; array[j2] = temp; }
      }
    sprintf(file, shell, gap);
    gpout(file, array, size);
  }
}
```

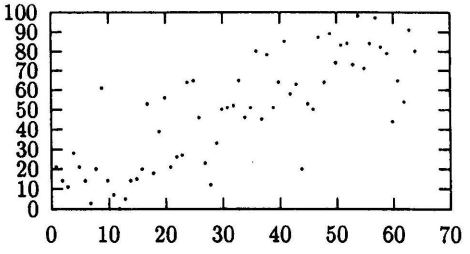
図 4: シェルソートのプログラム



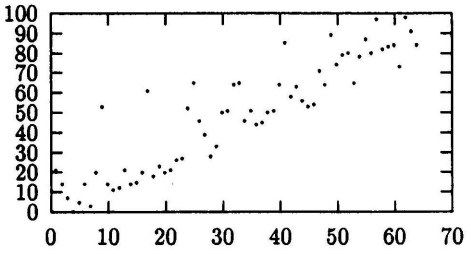
初期データ



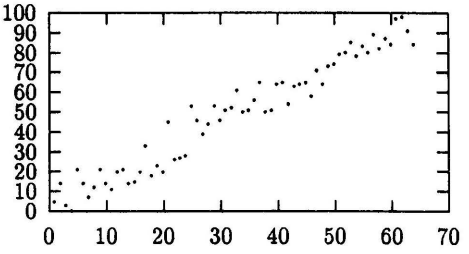
gap = 32



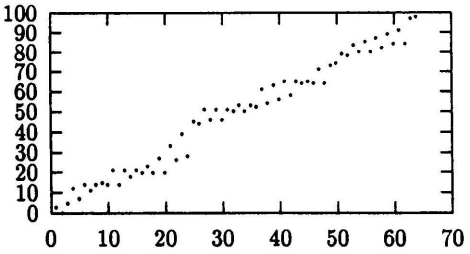
gap = 16



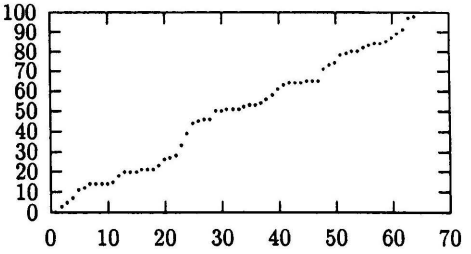
gap = 8



gap = 4



gap = 2



gap = 1

図 5: シェルソートの実行過程