Plots in LaTeX: `Gnuplot`, `Octave`, `make`

Boris Veytsman [*]   Leyla Akhmadeeva[†]

TUG2013

[*] Systems Biology School & Computational Materials Science Center, MS 6A2, George Mason University, Fairfax, VA, 22030, USA

[†] Bashkir State Medical University, 3 Lenina Str., Ufa, 450000, Russia

1. Goals

> This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.
>
> *Doug McIlroy*

1. We do not want to held computer’s arm. Computer should know what to do and when!
2. Harmony between the text and the plots. Same fonts, same style.
3. We want TeX labels on the plots.
4. We want to use external programs well designed to handle graphics.

1. Goals

> This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.
>
> *Doug McIlroy*

1. We do not want to held computer’s arm. Computer should know what to do and when!
2. Harmony between the text and the plots. Same fonts, same style.
3. We want TEX labels on the plots.
4. We want to use external programs well designed to handle graphics.

1. Goals

> This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.
>
> *Doug McIlroy*

1. We do not want to held computer's arm. Computer should know what to do and when!

2. Harmony between the text and the plots. Same fonts, same style.

3. We want TEX labels on the plots.

4. We want to use external programs well designed to handle graphics.

1. Goals

> This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.
>
> *Doug McIlroy*

1. We do not want to held computer’s arm. Computer should know what to do and when!
2. Harmony between the text and the plots. Same fonts, same style.
3. We want TEX labels on the plots.
4. We want to use external programs well designed to handle graphics.

1. Goals

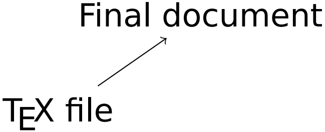
> This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.
>
> *Doug McIlroy*

1. We do not want to held computer's arm. Computer should know what to do and when!
2. Harmony between the text and the plots. Same fonts, same style.
3. We want TEX labels on the plots.
4. We want to use external programs well designed to handle graphics.

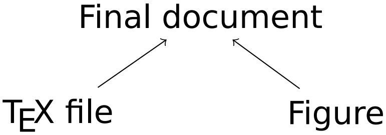
2. Makefiles

Final document

2. Makefiles



2. Makefiles

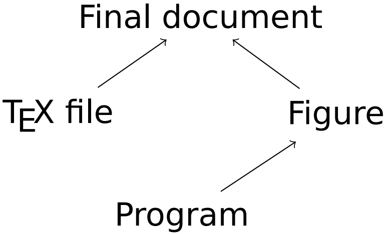


Final document

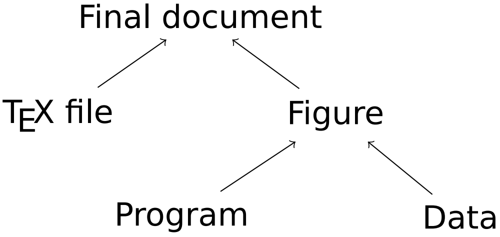
TEX file

Figure

2. Makefiles



2. Makefiles



Final document

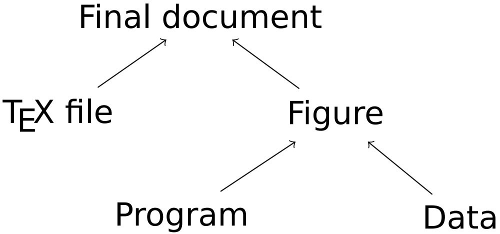
TEX file

Figure

Program

Data

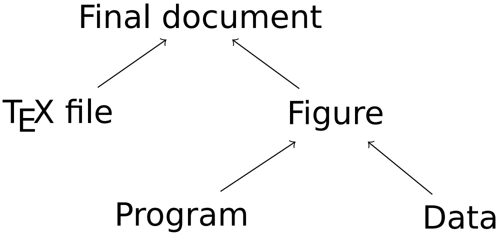
2. Makefiles



Dependencies:

1. If TEX file or figure change, we want to recompile the document.
2. If data or program change, we want to recompile the figure.

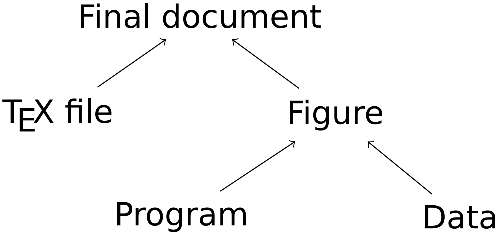
2. Makefiles



Dependencies:

1. If TEX file or figure change, we want to recompile the document.
2. If data or program change, we want to recompile the figure.

2. Makefiles



Final document

TeX file

Figure

Program

Data

Dependencies:

1. If TeX file or figure change, we want to recompile the document.

2. If data or program change, we want to recompile the figure.

Makefile & dependencies:

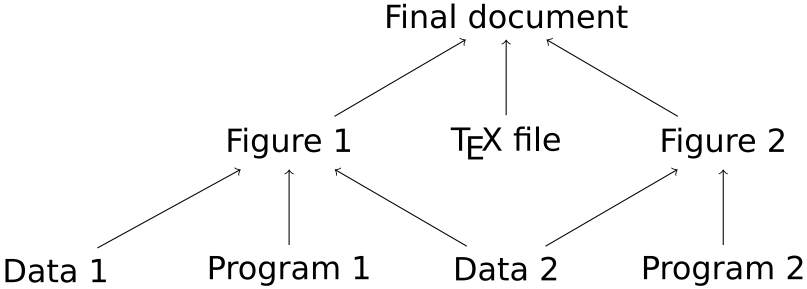
```
document.pdf: document.tex

document.pdf: figure-fig.tex

figure-fig.tex:  data.dat

figure-fig.tex:  figure.gp
```

A more complex case:



Final document

Figure 1

TEX file

Figure 2

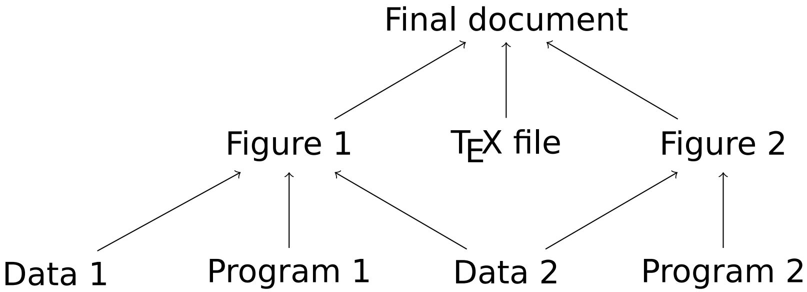
Data 1

Program 1

Data 2

Program 2

A more complex case:



```
document.pdf: document.tex figure1-fig.tex figure2-fig.tex

figure1-fig.tex:  data1.dat figure1.gp

figure2-fig.tex:  data1.dat data2.dat figure2.gp
```

Rules. How to make a PDF?

Rules. How to make a PDF?

```
%.pdf:  %.tex
    pdflatex $*
    pdflatex $*
    pdflatex $*
```

Rules. How to make a PDF?

```
%.pdf:  %.tex
    pdflatex $*
    pdflatex $*
    pdflatex $*
```

A smarter rule:

```
%.pdf:  %.tex
    pdflatex $*
    while ( grep -q \
       '^LaTeX Warning: Label(s) may have changed' $*.log ); \
       do pdflatex $*; \
    done
    pdflatex $*
```

3. TEX-compatible Graphics

1. A graphics program should generate a TEX file for textual material. . .

2. And a graphics file (EPS or PDF) to be included.

3. TEX-compatible Graphics

1. A graphics program should generate a TEX file for textual material. . .

2. And a graphics file (EPS or PDF) to be included.

3. TEX-compatible Graphics

1. A graphics program should generate a TEX file for textual material. . .

2. And a graphics file (EPS or PDF) to be included.

3. TEX-compatible Graphics

1. A graphics program should generate a TEX file for textual material. . .
2. And a graphics file (EPS or PDF) to be included.

In main TEX file:

```
\input{figure-fig}
```

3. TEX-compatible Graphics

1. A graphics program should generate a TEX file for textual material. . .
2. And a graphics file (EPS or PDF) to be included.

In main TEX file:

```
\input{figure-fig}
```

In Makefile

```
document.pdf:  figure1-fig.tex figure2-fig.tex ...

%-fig.tex:  DEPENDENCIES
    RULES
```

4. Gnuplot

Skeleton Program:

```
set terminal epslatex
set output ”FILE-fig.tex”
COMMANDS
set output
```

4. Gnuplot

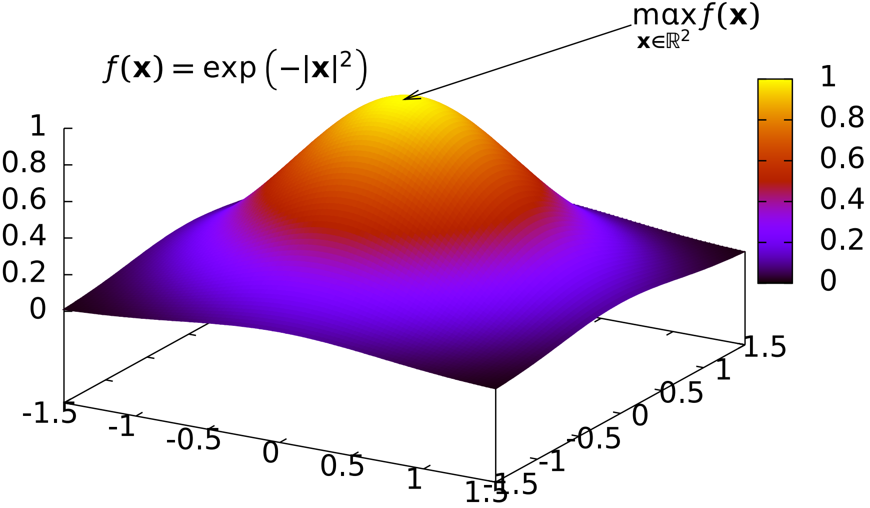
Skeleton Program:

```
set terminal epslatex
set output ”FILE-fig.tex”
COMMANDS
set output
```

Makefile:

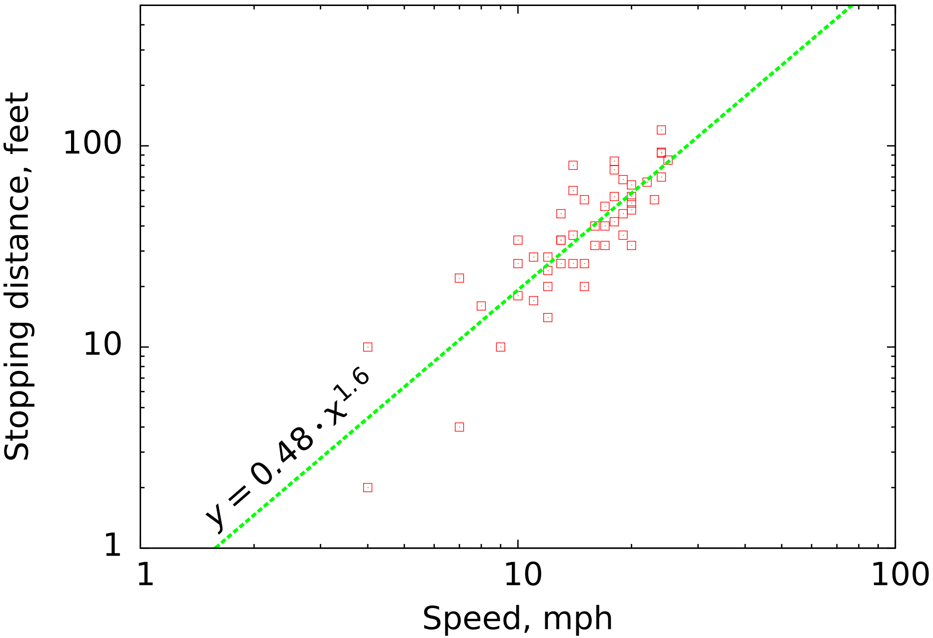
```
%-fig.tex:  %.gp
    gnuplot $<
```

Example:


$\max_{\mathbf{x} \in \mathbb{R}^2} f(\mathbf{x})$
$f(\mathbf{x}) = \exp\left(-|\mathbf{x}|^2\right)$
1
0.8
0.6
0.4
0.2
0
-1.5
-1
-0.5
0
0.5
1
1.5
-1.5
-1
-0.5
0
0.5
1
1.5
1
0.8
0.6
0.4
0.2
0

```
set terminal epslatex color
set output ”function-fig.tex”
set pm3d                # Colored surface
unset surface           # We do not want to plot the mesh lines
set isosamples 100, 100 # Smooth surface
set ztics 0.2           # Increment for z tick marks
set cbtics 0.2          # Increment for colored box
set xrange [-1.5:1.5]
set yrange [-1.5:1.5]
set label 1 \
 '$f(\mathbf{x})=\exp\left(-\lvert\mathbf{x}\rvert^2\right)$' \
 at -1.5,-1,1.2
set label 2 \
 '$\displaystyle\max_{\mathbf{x}\in \mathbb{R}^2} f(\mathbf{x})$' \
 at 1,1,1.3
set arrow 1 from 1,1,1.3 to 0,0,1 front
splot exp(-x**2-y**2) title ””
set output
```

Another example:



```
set terminal epslatex color
set output ”cars-fig.tex”
set logscale xy
set xrange [1:100]
set yrange [1:500]
set xlab 'Speed, mph'
set ylab 'Stopping distance, feet'
set label 1 \
  '\rotatebox{41}{$y=0.48\cdot x^{1.6}$}' \
  at 1.4, 3
plot ”cars.dat” with points pointtype 4 title ””, \
     exp(-0.73+1.6*log(x)) \
     linecolor 2 linewidth 5 title ””
set output
```

5. Octave

Skeleton program:

```
figure('visible','off');
COMMANDS
print -depslatex  ”-SX,Y” ”figure-fig.tex”
```

5. Octave

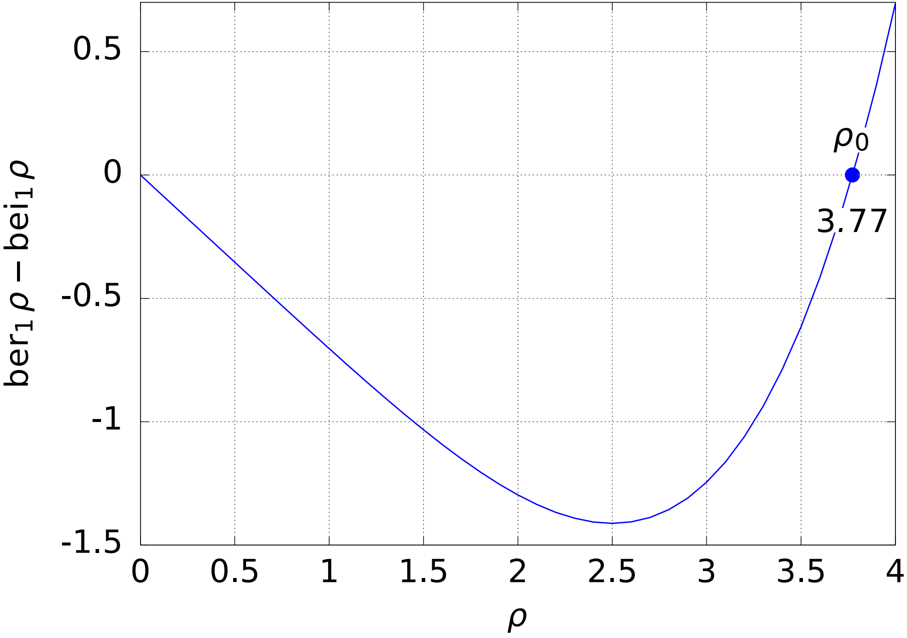
Skeleton program:

```
figure('visible','off');
COMMANDS
print -depslatex  "-SX,Y" "figure-fig.tex"
```

Makefile:

```
%-fig.tex:  %.m
    octave $<
```

Example:


0.5
0
-0.5
-1
-1.5
0
0.5
1
1.5
2
2.5
3
3.5
4
ρ
$\mathrm{ber}_1\,\rho - \mathrm{bei}_1\,\rho$
ρ_0
3.77

```
figure('visible','off');
ber1 = @(x) -real(besselj(1,x*exp(pi*1i/4)));
bei1 = @(x) imag(besselj(1,x*exp(1i*pi/4)));
delta = @(x) ber1(x)-bei1(x);
rho0 = fsolve(delta,4);
x=0:0.1:4;
plot(x,delta(x),'linewidth',2);
hold on;
plot([rho0], [0], 'o', 'linewidth', 10);
text(rho0, 0.15, '\colorbox{white}{$\rho_0$}', \
     'horizontalalignment', 'center');
text(rho0, -0.2, \
     sprintf("\\colorbox{white}{$%.2f$}", rho0), \
     'horizontalalignment', 'center');
title ("");  legend ("off");  grid();
xlabel('$\rho$');
ylabel('$\ber_1\rho-\bei_1\rho$');
print -depslatex "-S600,400" "kelvin-fig.tex"
```

```
figure('visible','off');
ber1 = @(x) -real(besselj(1,x*exp(pi*1i/4)));
bei1 = @(x) imag(besselj(1,x*exp(1i*pi/4)));
delta = @(x) ber1(x)-bei1(x);
rho0 = fsolve(delta,4);
x=0:0.1:4;
plot(x,delta(x),'linewidth',2);
hold on;
plot([rho0], [0], 'o', 'linewidth', 10);
text(rho0, 0.15, '\colorbox{white}{$\rho_0$}', \
     'horizontalalignment', 'center');
text(rho0, -0.2, \
     sprintf(”\\colorbox{white}{$%.2f$}”, rho0), \
     'horizontalalignment', 'center');
title ("");  legend (”off”);  grid();
xlabel('$\rho$');
ylabel('$\ber_1\rho-\bei_1\rho$');
print -depslatex ”-S600,400” ”kelvin-fig.tex”
```

Why this file would cause TeX errors?

Two macros: `\bei` and `\ber`. Need to define them (amsmath):

```
\DeclareMathOperator{\ber}{ber}
\DeclareMathOperator{\bei}{bei}
```

Two macros: `\bei` and `\ber`. Need to define them (amsmath):

```
\DeclareMathOperator{\ber}{ber}
\DeclareMathOperator{\bei}{bei}
```

Our generated TEX file uses fonts and macros from the main one!

6. Questions and Answers

Question: Gnuplot and Octave use EPS, but we use `pdflatex`. How does it work?

Answer: Modern TEX translates EPS graphics to PDF on the fly—and uses timestamps like `make`!

6. Questions and Answers

Question: Gnuplot and Octave use EPS, but we use `pdflatex`. How does it work?

Answer: Modern TEX translates EPS graphics to PDF on the fly—and uses timestamps like `make`!

6. Questions and Answers

Question: Gnuplot and Octave use EPS, but we use `pdflatex`. How does it work?

Answer: Modern TEX translates EPS graphics to PDF on the fly—and uses timestamps like `make`!

Question: It is too boring to write all these dependencies: `document.pdf: figure1-fig.tex figure2-fig.tex ...` Can computer do this for us?

Answer: Just use a script `makefigdepend.pl` and add to Makefile

6. Questions and Answers

Question: Gnuplot and Octave use EPS, but we use `pdflatex`. How does it work?

Answer: Modern TEX translates EPS graphics to PDF on the fly—and uses timestamps like `make`!

Question: It is too boring to write all these dependencies: `document.pdf: figure1-fig.tex figure2-fig.tex ...` Can computer do this for us?

Answer: Just use a script `makefigdepend.pl` and add to Makefile

6. Questions and Answers

Question: Gnuplot and Octave use EPS, but we use `pdflatex`. How does it work?

Answer: Modern TEX translates EPS graphics to PDF on the fly—and uses timestamps like `make`!

Question: It is too boring to write all these dependencies: `document.pdf: figure1-fig.tex figure2-fig.tex ...` Can computer do this for us?

Answer: Just use a script `makefigdepend.pl` and add to Makefile

```
depend:  ${TEXFILES}
   perl makefigdepend.pl \
   ${TEXFILES} > depend

-include depend
```

Question: What about cleaning the intermediate files?

Answer: Use `clean` goal:

Question: What about cleaning the intermediate files?

Answer: Use `clean` goal:

Question: What about cleaning the intermediate files?

Answer: Use `clean` goal:

```
clean:
   $(RM) *.aux *.bbl *.dvi *.log *.nav *.snm \
   *.out *.toc *.blg *.lof *.lot \
   *.eps *-pics.* *-fig* depend

distclean: clean
   $(RM) ${PDFS}
```

7. Conclusions

1. You can make a good scientific & engineering graphics with tools like Gnuplot and Octave

2. You can automate boring parts of your work with Makefiles

7. Conclusions

1. You can make a good scientific & engineering graphics with tools like Gnuplot and Octave
2. You can automate boring parts of your work with Makefiles

> Machines should work. People should think
> *An old IBM phrase*

A. Makefile for This Talk

```
TEXFILES = \
        gnuplotmk.tex

PDFS = ${TEXFILES:%.tex=%.pdf}

all: ${PDFS}

%.pdf:  %.tex
        $(RM) $*.toc
        pdflatex $*
        - bibtex $*
        $(RM) $*.toc
        pdflatex $*
        - while ( grep -q '^LaTeX Warning: Label(s) may have changed' $*.log ); \
        do pdflatex $*; done
        pdflatex $*

%-fig.tex: %.gp
        gnuplot $<
```

```
%-fig.tex: %.m
        octave $<

figure-fig.tex:
        touch $@

cars-fig.tex: cars.dat

clean:
        $(RM) *.aux *.bbl *.dvi *.log *.nav *.snm \
        *.out *.toc *.blg *.lof *.lot \
        *.eps *-pics.* *-fig* depend

distclean: clean
        $(RM) ${PDFS}



depend:  ${TEXFILES}
        perl makefigdepend.pl \
        ${TEXFILES} > depend

-include depend
```

B. Makefigdepend Script

```
#!/usr/bin/perl

#
# Extract information from input statements in TeX file
#
# Usage:
# makefigdepend FILE FILE FILE ... > depend
#

foreach my $file (@ARGV) {
    open FILE, $file;
    $file =~ s/\.tex$/.pdf/;
    while (<FILE>) {
        while (/\\input(?:\[[^\]]+\])*\{([^\}]+)\}/g) {
            print ”$file: $1.tex\n”;
        }
    }
    close FILE;
}
exit 0;
```