Appendix A: `fit`

`fit` is a non-linear least-squares fitting program that allows you to fit (at most) nine parameters of a function to data contained in a file. The function must be expressed as a single line of intrinsic Fortran functions, real numbers, operations (`+, -, *, /, **` or `^`) and adjustable parameters `K1-K9`. Both `plot` and `fit` use the prompt "`*`" to request commands for action from the user.

A typical `fit` session begins by using the editor to create a data file. When creating the data file, follow this convention: reserve the first line of the file for an identifying sentence, and follow that line with columns of data separated by tabs or spaces. Typically you will put x values in the first column, y values in the second column, and y-errors (if available) in the third column. `fit` assumes this file is called `fit.dat`, but other filenames may be `SET`.

Once you have created the file, start the program by typing `fit` to the *linux* `%` prompt:

```
% fit
*
```

(At any time you may exit `fit` by holding down the *Ctrl* key and simultaneously hitting *D*. **Note:** the common windoze command *Ctrl* C kills programs rather than copies; simply highlight by sweeping with mouse and then hit the center mouse button to paste at the desired location.) `SET fit`'s `FILE` variable equal to the *filename* of the file you just created by typing:

```
* SET FILE=filename
```

If you have no y-errors, inform the program by typing

```
* SET YECOL=0
```

(Don't forget to hit the *Return* or *Enter* key after each line. Note: you cannot edit these lines using arrow keys: you must backspace!)

If any other columns are out of place, re`SET fit`'s other column location variables: `XCOL` and `YCOL`. Note that `fit` will use no more than 512 data points. When `fit` is properly informed about your file type:

```
* READ
```

and `fit` will read your data into its internal data storage area.

The next step is to tell `fit` the function you want it to fit to your data. As stated above, you must be able to express this function in one line using the intrinsic Fortran functions (e.g., `log, log10, asin, ...`), constant real numbers (e.g., `2, 6.03e23`), operations (`+, -, *, /, **` or `^`), and (possibly) varying parameters (`K1-K9`). Examples:

```
* set f(x)=k1*exp(-((x-k2)/k3)**2)

* set f(x)=k1*(1-exp(k2*x))

* set f(x)=k1/sqrt((x-k2)^2+k3^2)
```

The function should be typed *without* using spaces. Note that the program automatically takes the absolute value of negative numbers raised to a power (e.g., $(-2)^3 = 8$), but the `SQRT` of a negative number is zero. Use repeated multiplications to simulate integer powers (e.g., $x^3 = \mathtt{x}*\mathtt{x}*\mathtt{x}$), if the proper sign is required. Note that the default `f(x)` is the polynomial $\mathtt{k1}+\mathtt{k2}*\mathtt{x}+\mathtt{k3}*\mathtt{x}^2+\cdots+\mathtt{k9}*\mathtt{x}^8$, defined with repeated multiplication.

Now it's time to `SET` your initial guess for the `K` parameters that occur in your function, with a command like:

```
* SET K1=3.14159 K2=4
```

Unlike linear least squares programs like *WAPP*$^+$you must provide a moderately accurate initial guess for each adjustible parameter. Time spent in careful consideration of parameter values will be repaid many-fold. Bad initial parameter guesses will result in program crashes, long computer `fit` times, and inaccurate computer results. `fit` will step into the first chi-square (χ^2) valley (χ^2 minimum) it finds, and then stay there; so you must start `fit` close to the right valley. It is advisable to try several different starting guesses for your parameters to find the path that leads to the lowest value of χ^2.

Once you have a good initial guess for your `K` parameters, you may ask the computer to iterate to a minimum χ^2. Of course, you must also tell the computer what `K` parameters it should vary to find that minimum:

```
* fit

Enter list of Ks to vary, e.g.  K1-K3,K5:
```

After you tell the computer what `ks` to vary, the machine will start its search. How does `fit` know when to stop? It will, of course, never reach the exact valley bottom (i.e., χ^2 minimum). If `fit` "`finished`" there is some evidence of convergence: either χ^2 changed hardly at all during its last downward step ($\Delta\chi^2$=`DELCHI`, default=.1), or the parameters' first `KSIG` digits (default=4) didn't change during its last downward step. Note: unchanging digits for one step, should not be interpreted as unchanging digits for lots more steps, nor should it be interpreted as an error. If `fit` "`quit`" then it's sick of trying (`SET` by `NITER`, default=100), so you should assume more steps are needed to reach the valley bottom. This suggests something is wrong. . . your initial guesses, the function, the data.

Once `fit` has quit or finished, it will print out summary results. Longer versions of the results (including the covariance matrix needed to figure errors[1]) may be printed, typed on the screen or placed in a file, at the users option. Always retain a copy of successful fit results! Sometimes the found χ^2 is so "bad" that you will be tempted to `FUDGE`, i.e., adjust your y-errors until you get a reduced-χ^2 of one. If you `FUDGE` be sure to write down why you think this cheating was justified.

A good way to estimate errors is to repeat the experiment lots of times. However very often the required time is not available. `fit` provides a possibility legitimate way to simulate actual repeats: it repeatedly fits using random subsets of your data. This process is called bootstrapping and it is invoked by the command `BOOTST`.

The command

```
    * PRINT
```

will display your data (x, y, y-error, and $f(x)$) on the screen. `fit` will mark (?, !, :) points that are 3, 2, or 1 y-errors away from $f(x)$. Use this feature to see which data points are poorly fit. Alternatively, you may use the program `plot` to display your data with fitted curve.

You may find it helpful to do some parameter stepping yourself before you hand the final fit over to `fit`. Put in a parameter guess, enter

```
    * CHI
```

to find the resulting χ^2, modify your guess and repeat the process.

`fit`ting functions to data is an important and frustrating part of your education in physics. Good luck!

Report problems and/or suggestions to Tom Kirkman.

[1]If the reduced χ^2 suggests a reasonable fit, the parameter uncertainties may be calculated from the square root of the corresponding diagonal element of the covariance matrix.

Special Commands

Start—in a terminal type: `fit`
`$`*linux* command — do any *linux* command from inside `fit`
`@`*filename* — execute indirectly a file of `fit` commands
Ctrl D — to exit program
Examples:

- `$ls *.dat`
- `@test.fit`

`fit` Commands

(unambiguous abbreviations and lowercase allowed)

Syntax:

- `COMMAND`
- `COMMAND VARIABLENAME =` value`, VARIABLENAME =` value `...`

Examples:

- `SET K1=0, K2=1, K3=-1, F(X)=K1+K2*X**2+K3*X**4`
- `re yec=0, fi=expt.dat`

`BOOTST`	performs a bootstrap with `NBOOT` iterations
`CHI`	calculates χ^2 using `F(X), Ks`, and data.
`FIT`	fits `F(X)` to your data (requests list of `Ks` to vary)
`FUDGE`	destroys existing y-errors and renorms them so $\chi^2 =$ `NPOINT`
`HELP`	displays these messages
`PRINT`	displays data stored in program and `F(X)` (marks large-deviation points)
`READ`	reads data from `FILE`: see `XCOL, YCOL, YECOL, ROW, IBEGIN, NPOINT, FILE`
`SET`	allows you to set variables values without doing any action
`SHOW`	shows variable values
`WRITE`	writes x, y, y-error, and $f(x)$ to `FILE`

`fit` Variables

`A`	constant for use in `F(X)` (0)
`B`	same (Note: `A` and `B` are used in `WINDOW(X)`: `WINDOW(X)=1` iff $a \leq x \leq b$, else zero)
`C`	same
`CHISQ`	most recent value of χ^2 calculated
`DELCHI`	`fit` stops if a step results in a χ^2 change less than `DELCHI` (.1)
`F(X)`	function use: `A, ABS, ACOS, ASIN, ATAN, B, C, COS, COSH, EXP, INORM, K, K1-K9, LOG, LOG10, NORM, PI, SIN, SINH, SQRT, TAN, TANH, WINDOW, X, any real number, operations +, -, *, /, **,` or `^` (`K1+X*(K2+X*(K3+X*(K4+X*(K5+X*(K6+X*(K7+X*(K8+X*K9)))))))`)
`FILE`	`READ/WRITE`s data with this filename (`fit.dat`)
`IBEGIN`	first array element used during `CHI, FIT, READ, PRINT, WRITE` (1)
`K1-K9`	possibly adjustable variables in `F(X)`
`KSIG`	`fit` stops after a step if all `K`s retain `KSIG` unchanged digits (4)
`NBOOT`	number of bootstrap iterations performed (25)
`NITER`	`fit` stops if more than `NITER` test steps are needed (100)
`NPOINT`	number of points used during `CHI, FIT, READ, PRINT, WRITE` (512)
`ROW`	first row of data in `FILE` (2)
`SEED`	if negative, re-initializes random generator's seed (-1)
`XCOL`	column of x data in `FILE` — used during `READ` (1)
`YCOL`	column of y data in `FILE` — used during `READ` (2)
`YECOL`	column of y-error data in `FILE` — used during `READ` (3) Note: unread y-errors default to 1