m u l t i F I T

The program *multiFIT* (less than *800 bytes*) was written by *Bruno Hampel* for the *Hewlett-Packard HP 48GX & appropriate series* in the *middle of the nineties*, and it is *freeware*. You are permitted to use this program and its documentation in a non-commercial environment without any payment. The personal give-away is allowed, as far as you hand out the complete and unmodified package as well as you do not earn money doing this. Any other usage requires a separate written permission. The program is presented "as-is" and there is no warranty neither concerning the functionality of the algorithm, nor the delivered results.

As far as there is small resonance on it, you can try to contact me by e-mail at *bruno@cerasela.de* or at *bruno.hampel@t-online.de*. You can send me a fax at *+49 – 69 – 7 91 24 56 34*, too. I will do my best to answer you. On big resonance, I will establish a support site at *www.cerasela.de*.

You can either use the *Serial Interface Kit* to transfer the file *mf* to your calculator (do not change anything in this file), or you can type in the program, which is listed at the end of this document.

In *"Introduction"* I will introduce the purpose of the program and the notifications used. There is a brief *"Mathematical Deduction"* following, where I am relating about the mathematical background. Then you can read about how to use my program in the section *"Program Usage"*. Under *"Examples"* you will encounter four examples. In *"Listing"* there is the complete code of the program listed.

Kaarst in Germany, the 13$^{\text{th}}$ of November 1999

Introduction

The program *multiFIT* can compute (method: *Least Squares Fitting*) the constants $c_k \in \mathbb{R}$ for an equation of the following appearance:

$$y\left(x_1, x_2, \ldots, x_m\right) = \sum_{k=1}^{n} c_k \cdot f_k\left(x_1, x_2, \ldots, x_m\right)$$

My program does a little bit more than a multiple regression – it performs a multi-dimensional regression. Unfortunately, the regression can be done for the liner c_k, only. However, this is much more than the calculator can do originally, particularly because there is just one limitation for the f_k among one another. The data $(z, x_1, x_2, \ldots, x_m)$, which you may have obtained previously by measurements, might look this way:

i	x_1	x_2	$\ldots$	x_m	z
1	$\langle x_1 \rangle_1$	$\langle x_2 \rangle_1$		$\langle x_m \rangle_1$	$\langle z \rangle_1$
2	$\langle x_1 \rangle_2$	$\langle x_2 \rangle_2$		$\langle x_m \rangle_2$	$\langle z \rangle_2$
$\vdots$					$\vdots$
s	$\langle x_1 \rangle_s$	$\langle x_2 \rangle_s$	$\ldots$	$\langle x_m \rangle_s$	$\langle z \rangle_s$

Please get used to my notifications. The f_k are any functions depending on none, one or even more variables x_j, where $j \in \{1, 2, \ldots, m\}$. This dependency on variables can be independent for the different f_k. There should exist $s > n$ data sets, each consisting of $m + 1$ values. The z refers to the measured value for y, while y itself refers to the potentially calculated value, which would be exact if we knew the c_k exactly. In the table above $\langle x_j \rangle_i$ is the abbreviation for the i^{th} value of the j^{th} variable. The abbreviation $\langle f_\lambda \rangle_i$ is equivalent to $f_\lambda\left(\langle x_1 \rangle_i, \langle x_2 \rangle_i, \ldots, \langle x_m \rangle_i\right)$.

Mathematical Deduction

This deduction bases on the assumption that the measured values $\langle \mathbf{x_j} \rangle_\mathbf{i}$ are all exact and the tolerances $\langle \mathbf{w} \rangle_\mathbf{i} = \mathbf{w}$ are for all $\langle \mathbf{z} \rangle_\mathbf{i}$ equal. The difference $|\langle z \rangle_i - \langle y \rangle_i|$ is made between measured value $\langle z \rangle_i$ and calculated (exact) value $\langle y \rangle_i$. Of course, $\langle y \rangle_i$ is not valuable because we do not know the values for the c_k – that is what we are going to do.
The probability to obtain a definite value $\langle z \rangle_i$ can be described by the *Gaussian Distribution* using next proportionality:

$$P_w\left[\langle z \rangle_i\right] \sim \exp\left\{-\frac{1}{2 \cdot w^2} \cdot \left[\langle z \rangle_i - \langle y \rangle_i\right]^2\right\}$$

The parameter w is called *width parameter* of the *Gaussian Curve*. The standardizing factor (constant and w-dependent) is of no interest for us because we are finally interested in the exponent.
I shortly introduce now the equation for w in order to clarify, where the condition $s > n$ does come from:

$$P \equiv P(w) \quad and \quad \frac{\partial P}{\partial w} = 0 \quad \Rightarrow \quad \ldots \quad \Rightarrow \quad w^2 = \frac{1}{s-n} \cdot \sum_{i=1}^{s} \left[\langle z \rangle_i - \langle y \rangle_i\right]^2 \geq 0 \quad \Rightarrow \quad s > n$$

Do not match this result into the equation for the probability P_w! **As we see, the required number of measurements should be greater than the number of coefficients!** For example if you would like to fit $y(x) = b + m \cdot x$, which contains two coefficients, you must have measured at least three (y, x)-pairs. Otherwise, through two points, it is always possible to draw a straight line. This kind of reflection can be extended for multi-dimensional surfaces, which require a minimum of points to be defined by.
The probability to obtain the whole set of values $\{\langle z \rangle_1, \langle z \rangle_2, \ldots, \langle z \rangle_s\}$ is then given by:

$$P_w\left[\langle z \rangle_1, \langle z \rangle_2, \ldots, \langle z \rangle_s\right] = \prod_{i=1}^{s} P_w\left[\langle z \rangle_i\right]$$

Introducing the next definition will simplify the notifications:

$$q \underset{def}{=} \frac{1}{w^2} \cdot \sum_{i=1}^{s} \left[\langle z \rangle_i - \langle y \rangle_i\right]^2$$

It would have been more instructive than simple to define q^2 instead of q because the computations used are called *least squares fitting* and this name comes from the q^2. However, using the definition given above, the probability becomes:

$$P_w\left[\langle z \rangle_1, \langle z \rangle_2, \ldots, \langle z \rangle_s\right] \sim e^{-\frac{1}{2} \cdot q}$$

Our intention is to compute the c_k in a way that will cause the probability to reach a maximum. Therefore we regard to the q as being a function of the c_k:

$$q \equiv q(c_1, c_2, \ldots, c_n)$$

In order to maximize P_w, we have to minimize q. That is why the next two conditions must be fulfilled:

$$\frac{\partial}{\partial c_\lambda} q = 0 \quad and \quad \frac{\partial^2}{\partial c_\lambda^2} q > 0 \quad for\ all \quad \lambda \in \{1, 2, \ldots, n\}$$

The first derivative is:

$$\frac{\partial q}{\partial c_\lambda} = -\frac{2}{w^2} \cdot \left\{ \sum_{i=1}^{s} \langle z \rangle_i \cdot \langle f_\lambda \rangle_i - \sum_{i=1}^{s} \sum_{k=1}^{n} c_\lambda \cdot \langle f_k \rangle_i \cdot \langle f_\lambda \rangle_i \right\}$$

Setting the brace to zero will deliver:

$$\sum_{i=1}^{s} \langle z \rangle_i \cdot \langle f_\lambda \rangle_i = \sum_{k=1}^{n} \sum_{i=1}^{s} c_\lambda \cdot \langle f_k \rangle_i \cdot \langle f_\lambda \rangle_i \quad for\ all \quad \lambda \in \{1, 2, \ldots, n\}$$

These represent n equations, each containing n variables. The result can be written as:

$$B = M \cdot X$$

Here is $M = M^T$ a symmetric $n \times n$ matrix. B and X are both n-element vectors:

$$M_{\alpha\beta} = \sum_{i=1}^{s} \langle f_\alpha \rangle_i \cdot \langle f_\beta \rangle_i \quad and \quad B_\beta = \sum_{i=1}^{s} \langle z \rangle_i \cdot \langle f_\beta \rangle_i \quad and \quad X = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}$$

We will obtain the values for the $\mathbf{c_k}$ if we solve the equation $\mathbf{B = M \cdot X}$. My program delivers B and M. At this point, we know how the c_k have to be computed causing the probability be an extreme.
To fix if this extreme is the required minimum, we take a look to the second derivation:

$$\frac{\partial}{\partial c_\lambda} \frac{\partial q}{\partial c_\lambda} = \quad \ldots \quad = \frac{2}{w^2} \cdot (-1)^2 \cdot \frac{\partial}{\partial c_\lambda} \sum_{k=1}^{n} \sum_{i=1}^{s} c_\lambda \cdot \langle f_k \rangle_i \cdot \langle f_\lambda \rangle_i = \frac{2}{w^2} \cdot \sum_{i=1}^{s} \langle f_\lambda \rangle_i^2 > 0$$

As we can see, the sum of squares is always greater than zero, so that if we calculate the c_k using $B = M \cdot X$, the q gets its minimum and therefore P_w is maximized.
You surely remarked that my program does not deliver X but B and M, and you may ask yourself why Bruno (that's me) did not put this final step in his program. Well, if the resulting matrix were a matrix with $\det(M) = 0$ or one that is of bad condition, the final division would cause an error. I could trap such an error but this would increase the size of the program considerably and you would not know what is happening. By the way, I did not trap any possible error! To avoid matrices with $\det(M) = 0$ you should take care that the following condition is fulfilled:

$$f_a \neq \mu \cdot f_b \quad for\ all \quad a, b \in \{1, 2, \ldots, n\} \quad where \quad a \neq b \quad and \quad \mu \in \mathbb{R}$$

In words: **For different a and b it is necessary that $\mathbf{f_a}$ and $\mathbf{f_b}$ can be distinguished by more than a real number.**

Here two examples (you can use the substitute $\tilde{y}$ instead of y):

$$y_1(x,y) = a \cdot \underline{\sin(x)} + b \cdot \sin(y) + c \cdot \underline{\sin(x)} \quad \rightarrow \quad \tilde{y}_1(x,y) = \underbrace{c_1}_{a+c} \cdot \sin(x) + \underbrace{c_2}_{b} \cdot \sin(y)$$

Since you can compute c_1 and c_2, you can obtain the value for b but it is not possible to get a and c separately (this is a simple example, I know). The second example:

$$y_2(x,y,z) = a \cdot x^3 + c \cdot z + d \cdot \left(x^3 + z\right) \quad \Longleftrightarrow$$

$$y_2(x,y,z) = a \cdot \underline{x^3} + c \cdot \underline{z} + d \cdot \underline{x^3} + d \cdot \underline{z} \quad \rightarrow \quad \tilde{y}_2(x,y,z) = \underbrace{c_1}_{a+d} \cdot x^3 + \underbrace{c_2}_{c+d} \cdot z$$

In this example *multiFIT* can compute c_1 and c_2 but you will not be able to get the values for a, c and d. Using $\tilde{y}$ (necessary because of the condition on page four) may cause loss of some coefficients that have been initially used in y. See the fourth example for a complicated equation, too!
Bad conditioned matrices are those consisting in elements, which may convert to zero by internal calculations even if these elements are not really zero, but a very small number. That is why such a matrix can suddenly include at least one row or column composed of zeros. This behavior is due to the numeric conversion of the matrix and to the limitations concerning the representation of numbers in any computing machine. To learn more about bad conditioned matrices and how to operate with, please consult the manual of your calculator.

Program Usage

Do not understand most of the following brackets and braces as being math stuff but syntax of the calculator! There are three objects that need to be transferred to the stack before executing the program:

1. List of functions:

$\{ \quad f_1 \quad f_2 \quad \dots \quad f_n \quad \}$

This is a list containing all the f_k as algebraic elements of the list.

2. Data matrix:

$[\quad [\quad \dots \quad] \quad [\quad \dots \quad] \quad \dots \quad [\quad \dots \quad] \quad]$

This is a matrix containing the measured values. This one should have at least $m+1$ columns. Other columns are not relevant to this program. The number of rows is taken automatically for s – the number of measurements. Do not confuse the data matrix with the matrix M.

3. Specification list:

$\{ \quad column\,(z) \quad x_1 \quad column\,(x_1) \quad x_2 \quad column\,(x_2) \quad \dots \quad x_m \quad column\,(x_m) \quad \}$

This is a list of $1+2\cdot m$ elements, specifying where the program can find its values in the matrix given above. The first element is always a number related to the column where the measured value for the function is stored in. The following pairs of elements indicate in which column the named variable can be found in. The alignment of these pairs among one another is free. I introduced this list for purposes of facility – you should not extract the relevant columns initially.

4. Execute the multiFIT program:

There is no affection by any flags, and there will be no changes to your calculator after the program terminates (proper termination is not required). It will deliver two objects to the stack: the vector B and the matrix M:

$[\quad B_1 \quad B_2 \quad \dots \quad B_n \quad]$

$[\quad [\quad M_{11} \quad M_{12} \quad \dots \quad M_{1n} \quad] \quad \dots \quad [\quad M_{n1} \quad M_{n2} \quad \dots \quad M_{nn} \quad] \quad]$

5. Final steps:

The easiest way to get the final result is pressing the division key. You will obtain:

$[\quad c_1 \quad c_2 \quad \dots \quad c_n \quad]$

If there is an error occurring, the previously delivered matrix probably has been bad conditioned – see your manual for help (relevant are the command *LSQ* in dependency with *flag 22* and *flag 54*). Bad conditioned matrices may be very rarely. This is the reason why these operations have not accessed my program. Another source for an error could be negligence in setting up the fit equation. Do you remember the one and only condition that must be fulfilled?

Examples

Example 1: Imagine that you would like to fit $p(x)$ to the data (p, x) that you have obtained before by measurements and in order to that you want to get the values for a, b and c:

$$p(x) = a \cdot x^3 + b \cdot x^2 - c$$

Near to my notification, we identify:

$$f_1 \equiv x^3 \quad f_2 \equiv x^2 \quad f_3 \equiv -1 \quad c_1 \equiv a \quad c_2 \equiv b \quad c_3 \equiv c$$

1st step: Put the list of functions on the stack:

$$\{ \quad 'x\hat{}3' \quad 'x\hat{}2' \quad '-1' \quad \}$$

2nd step: Put the data matrix on the stack (however your matrix looks like).

3rd step: Put the specification list on the stack:

$$\{ \quad 4 \quad x \quad 2 \quad \}$$

In this example (this is just an example), the list above tells the program that p is located in the fourth column and x has its location in the second column.

4th step: Run the program. You will obtain a vector and a matrix as a two-object result on your stack.

5th step: Simply press the division key. You will obtain a vector containing the coefficients you searched for:

$$[\quad value(a) \quad value(b) \quad value(c) \quad]$$

Example 2: Imagine that you would like to fit $q_b(x, y, z)$ to the data (q_b, x, y, z) you have obtained before by measurements and in order to that you want to get the values for u, $(v-4)$ and w:

$$q_b(x, y, z) = u \cdot \left(x^3 \cdot z\right) + (v-4) \cdot (x \cdot y) + w \cdot \left(z^{5-b}\right)$$

For q_b the variable b is a parameter which cannot be computed by this program! You must know its value before starting the program.

$$f_1 \equiv x^3 \cdot z \quad f_2 \equiv x \cdot y \quad f_3 \equiv z^{5-b} \quad c_1 \equiv u \quad c_2 \equiv v-4 \quad c_3 \equiv w$$

The list of functions is (replace ? by the value for b or the name of the variable b used on your calculator):

$$\{ \quad '(x\hat{}3) * z' \quad 'x * y' \quad 'z\hat{}(5-?)' \quad \}$$

Put the data matrix on the stack. The specification list could be (the following one is randomized):

$$\{ \quad 2 \quad y \quad 6 \quad z \quad 5 \quad x \quad 3 \quad \}$$

Its meaning is: q_b is located in the second column, y is located in the sixth column, z is located in the fifth column and x is located in the third column. Finalizing the operations would deliver:

$$[\quad value(u) \quad value((v-4)) \quad value(w) \quad]$$

Example 3: Imagine that you would like to fit $r_a(t)$ to the data (r_a, t) obtained before by measurements and you want to get the values for A and B:

$$r_a(t) = A \cdot \sin(a \cdot t) - B \cdot \cos(a \cdot t)$$

The coefficients A and B can be computed. The parameter a has to be known before.

$$f_1 \equiv \sin(a \cdot t) \quad f_2 \equiv -\cos(a \cdot t) \quad c_1 \equiv A \quad c_2 \equiv B$$

The list of functions is:

$$\{ \quad \text{' } \sin(? \cdot t) \text{ ' } \quad \text{' } -\cos(? \cdot t) \text{ ' } \quad \}$$

Replace ? by the value for a or its name on your calculator. Send the data matrix to the stack and after that do your specifications:

$$\{ \quad 4 \quad t \quad 3 \quad \}$$

In this example (and only here), the list above would tell the program that r_a is located in the fourth column and the variable t is located in the third column. Run the program and execute the required division. The result will be:

$$[\quad value(A) \quad value(B) \quad]$$

Example 4: There are some functions that allow operating with, provided that you convert them to look similar to my very first definition $y = \sum_k c_k \cdot f_k$. As an example:

$$r(x,y,z) = \frac{D \cdot e^{A+B\cdot x+C\cdot y}}{2^{E\cdot z+F}} \quad \Longleftrightarrow \quad r(x,y,z) = \frac{D \cdot e^{A+B\cdot x+C\cdot y}}{e^{\ln(2)\cdot(E\cdot z+F)}}$$

Some elementary transformations considering $r > 0$ and $D > 0$ lead us to a shape that looks quite similar to the basic $y = \sum_k c_k \cdot f_k$:

$$\ln r = \ln\left(D \cdot e^{A+B\cdot x+C\cdot y}\right) - \ln\left(e^{\ln(2)\cdot(E\cdot z+F)}\right) \quad \Longleftrightarrow$$

$$\ln r = \underline{\ln(D)} + \underline{A} + B \cdot x + C \cdot y - \ln(2) \cdot E \cdot z - \underline{\ln(2) \cdot F}$$

Remember please the condition given on page four, which is $f_a \neq \mu \cdot f_b$. Due to this restriction, we have to collect all constants belonging to $f_1 \equiv 1$.

$$\ln r = \underbrace{[\ln(D) + A - \ln(2) \cdot F]}_{c_1} \cdot 1 + \underbrace{B}_{c_2} \cdot x + \underbrace{C}_{c_3} \cdot y - \underbrace{\ln(2) \cdot E}_{c_4} \cdot z$$

My program can compute c_1 through c_4, and therefore you can calculate B, C and E. You will not be able to get the values for A, D and F, except there are any other equation setting some relations between A, D and F. Of course, you must generate a column containing $\ln(r)$ prior to the usage of *multiFIT*.

Listing

As you can see, the program is very short. Do not take this code for an explanatory algorithm. It is self-evident that the pure algorithm is comprised in this code, but I widely made use of programming tricks, which are strongly tied to the capabilities of the calculator and its programming language. Therefore, this listing is rather a distorted algorithm than a platform-transferable.

```
<<
    DUP SIZE 1 - 2 / 4 ROLL DUP SIZE DUP DUP2 2 →LIST
    RANM 0 * ROT 1 →LIST RANM 0 * 7 ROLL DUP SIZE HEAD
    <<
        → a V m
        <<
            m 0
            IF >
            THEN
                1 m
                FOR i
                    a V i 2 * GET 'd(k,l)' 'l' V i 2 * 1 + GET 2
                    →LIST ↓MATCH DROP 2 →LIST ↓MATCH DROP 'a' STO
                NEXT
            END
            a
        >>
    >>
    0
    → V m f n A B D N M z
    <<
        V HEAD 'z' STO
        1 n
        FOR i
            A i i 2 →LIST k 1 N f i GET 2 ^ V m M EVAL
            { d D } ↓MATCH DROP Σ  PUT 'A' STO B i k 1 N f i
            GET 'd(k,z)' * V m M EVAL { d D } ↓MATCH DROP Σ PUT 'B' STO
        NEXT
        1 n 1 -
        FOR i
            i 1 + n
            FOR j
                'k' 1 N f i GET f j GET * V m M EVAL { d D } ↓MATCH DROP
                Σ 'z' STO A i j 2 →LIST z PUT j i 2 →LIST z PUT 'A' STO
            NEXT
        NEXT
        B A
    >>
>>
```