FITS WITH CONSTRAINTS

FITCST.TEX KB 20040111

KLAUS BETZLER[1], FACHBEREICH PHYSIK, UNIVERSITÄT OSNABRÜCK

This short note presents an additional aspect sometimes important during data fitting – constraints. I assume the reader is familiar with basic aspects of fitting as presented, e. g., in my short lecture note "Fitting in MATLAB". Here, we will mainly discuss constraints imposed on parameter-linear fits like, e. g., a fixed maximum position in a polynomial fit.

1 Mathematics of Constraints

As we have seen in the preceding note, a fit function f is defined for a fit which generally contains adjustable parameters, the *fit parameters* $a_i, i = 0 \ldots n$. This fit function is used to describe a set of data $y_k, k = 1 \ldots M$ which – in the simplest case – are collected as a function of *one* independent variable x at the points x_k. We would like the fit function f to describe these data in an approximative way as

$$y_k \approx f(a_0, \ldots a_n, x_k) \; . \tag{1}$$

To find the parameters a_i for the best approximation we usually minimize the sum over the squared residuals

$$r = \sum_k (y_k - f(a_0, \ldots a_n, x_k))^2 \; . \tag{2}$$

The minimum is found by forcing the derivatives of r with respect to all a_i to be zero

$$\frac{\partial r}{\partial a_0} = 0, \quad \frac{\partial r}{\partial a_1} = 0, \quad \ldots \; , \quad \frac{\partial r}{\partial a_n} = 0 \; . \tag{3}$$

Whether and how these partial derivatives can be calculated depends on the detailed form of the function f. In the preceding note we discussed how to use MATLAB for several classes of fit functions including polynomials, parameter-linear functions, and arbitrary functions. We could show that for the first two classes exact results for the fits can be calculated using the MATLAB methods `polyfit` and `mldivide` whereas for the third very general class of functions optimization schemes are to be used (e. g. `fminsearch`).

It is obvious that a fit approximates the individual data the better the more parameters are used. Constraints introduce global conditions into the fit which reduce the

[1] KLAUS.BETZLER@UOS.DE

number of free parameters. Thus a fit usually gets worse – concerning the sum of the squared residuals – when we introduce constraints. Yet it will better describe the facts which motivate the constraints, i. e. the physics behind the data.

Mathematically each constraint imposes an additional relation between the fit parameters to the equation system Eq. 3 describing the fit. In an algebraic approach one would introduce the constraint equations at a very early point into the fitting scheme thus reducing the number of equations in the system. For each constraint one would express one of the parameters in the fit function Eq. 1 as a function of the other ones and eliminate it from the system. In MATLAB it is not necessary to do this algebraic elimination at an early point, one may introduce the constraint just before solving the fit problem.

Example. Let's take a set of data, x_i and corresponding y_i, a noisy second order parabola with its minimum at $x_m = 4$:

```
rand('state',1);
x = linspace(0,8,15)';
y = 0.7*(x-4).^2 + 5*rand(size(x));
```

To describe the data, we want to use a second order polynomial

$$Y = a_1 + a_2 X + a_3 X^2 \,. \tag{4}$$

In MATLAB three statements will solve the problem

```
a = polyfit(x,y,2);
X = linspace(min(x),max(x),100);
Y = polyval(a,X);
```

We introduce the vector X with appropriately dense spacing in the second statement in order to get a smooth curve.

Now what if we know that the minimum of the parabola *must* be exactly at $x_m = 4$? A constraint is imposed on our fit

$$Y'(x_m) = 0 \quad \text{which means} \quad 0 \cdot a_1 + 1 \cdot a_2 + 2 \cdot a_3 \cdot x_m = 0 \,. \tag{5}$$

We have an additional condition which in principle reduces the number of fit parameters by one. We can account for this by introducing the constraint condition into Eq. 4.

The Function `polyfit` of course then is no longer suitable. However, we still have a *parameter-linear* fit which can be exactly solved using the *matrix left division* as described. Formally we can introduce the constraint as shown in the short sequence below. The left and right side of the constraint equation are multiplied by the appropriate column ($C = 3$) of the coefficient matrix and subtracted from the complete system. The solution is then done in the usual way.

```
C = 3;                          % column
xm = 4;
Cvec = [0,1,2*xm];              % left side
Cvec = Cvec/Cvec(C);            % normalize
Csca = 0/Cvec(C);               % right side
A = [x.^0,x.^1,x.^2];
A = A - A(:,C)*Cvec;
y = y - A(:,C)*Csca;
a = A\y;
a(C) = -Cvec*a;                 % calculate missing
Y = [X.^0,X.^1,X.^2]*a;
```

MATLAB will tell you that the matrix is rank deficient but nevertheless will solve it, setting a_2 to zero. After retrieving a_2 the new curve can be calculated. Fig. 1 shows the results of the unconstrained and constrained fits for a set of noisy data.

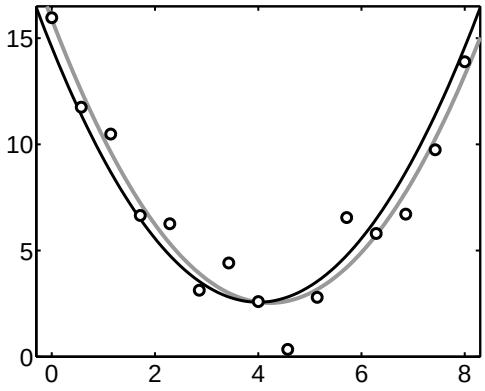

Figure 1: Second order polynomial fit to a set of noisy data. Dots denote the data points, the gray line represents an unrestricted fit, the black line a fit with the constraint that the minimum must be at $x = 4$.

2 Application: Liquidus and Solidus Curve

An application of such a fit with constraints is the phase diagram crystal growers use for determining the conditions under which crystals can be grown – compositions and temperatures. Two curves are important, the *liquidus* and the *solidus* curve. If a melt is cooled down, crystallization starts at a temperature defined by the liquidus curve. With a composition, however, which is defined by the solidus curve for the same temperature. In many cases of mixed crystals there is one special composition where a crystal grows at the melt composition. This is called the *congruently* melting composition. At this composition liquidus and solidus curve meet each other with a horizontal tangent. Crystal growers like these points as crystal growth is greatly facilitated there. Therefore these points for most systems are well known, always better than the rest of the liquidus and solidus curves. If one constructs these curves mathematically, the facts about the congruently melting compositions can thus be used as constraints.

As examples the liquidus and solidus curves of Strontium Barium Niobate are calculated. The experimental data for the liquidus curve were determined using a special technique of oscillatory growing and melting small crystal samples from different melts. The compositions of the crystals were measured by X-ray fluorescence analysis. Therefrom the data for the corresponding solidus curve were derived. The two data sets used for the fits are listed in Table 1.

x_l	0.194	0.3	0.431	0.547	0.569	0.61	0.7	0.805	0.85	0.9
T_l	1469.7	1479.6	1489.6	1491.1	1491.1	1491.6	1490.7	1488.8	1486.3	1485.3

x_s	0.322	0.383	0.477	0.563	0.612	0.689	0.789	0.822
T_s	1469.7	1479.6	1489.6	1491.1	1491.6	1490.7	1488.8	1486.3

Table 1: Liquidus and solidus temperatures T_l and T_s (in °C) for Strontium Barium Niobate of various compositions (Strontium content in the melt x_l and in the crystal x_s).

For the fit curves, fourth order polynomial were chosen, the maxima of them, i. e. the congruently melting composition, were forced to $x_c = 0.61$. Compared to the example shown in the preceding section, the constraint vector and the coefficient matrix have to be extended to account for fourth order in the MATLAB script.

```
Cvec = [0,1,2*xc,3*xc^2,4*xc^3];
```

and

```
A = [x.^0,x.^1,x.^2,x.^3,x.^4];
```

Figs. 2 shows the result for the liquidus curve. For comparison, an unconstrained fit curve is plotted, too. The maximum of the unconstrained fit is slightly shifted,

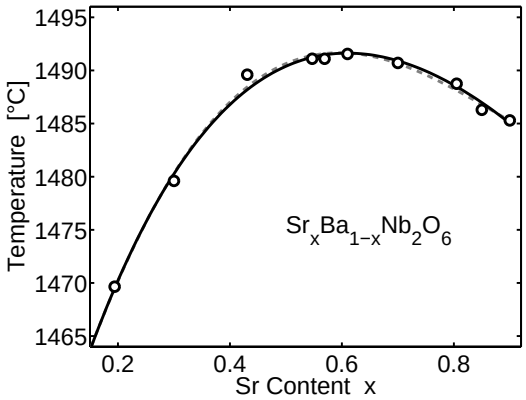

Figure 2: Liquidus curve of Strontium Barium Niobate. The dots denote the measured values, the gray dashed curve is an unconstrained fourth order polynomial fit to the data, the black solid curve is a fit with the maximum forced to $x = 0.61$.

at $x = 0.597$, and the sum of the squared residuals is slightly less (2.87 compared to 3.27 for the constrained fit).

Fig. 3 shows the result for the solidus data. Here the difference between constrained and unconstrained fit is hardly visible. Again, the maximum is slightly shifted, to $x = 0.605$, and the sum of the squared residuals is slightly less (1.72 compared to 1.74).

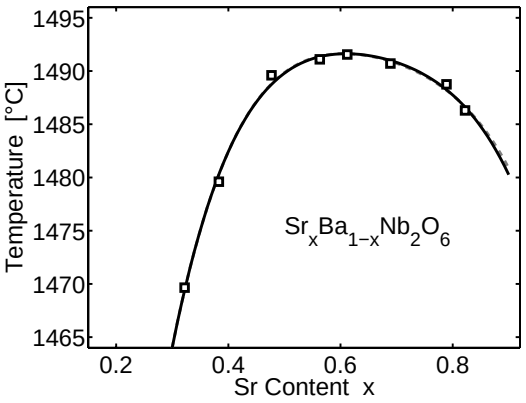

Figure 3: Solidus curve of Strontium Barium Niobate. The dots denote the measured values, the black solid curve is a fourth order polynomial fit to the data with the maximum forced to $x = 0.61$. The gray dashed curve again represents the unconstrained fit, the difference, however, is hardly detectable.

3 Multiple Constraints: The Phase Diagram

To get the complete phase diagram, we have to introduce an additional constraint – the temperatures of the liquidus and the solidus curve at the congruently melting composition must be equal. Therefore we have to calculate a parameter-linear fit for the whole thing with three constraints.

In such a case of multiple constraints a recursive approach should be used. The constraints are introduced, one after each other. Then the problem is solved by matrix left division. The missing parameters then are to be inserted in reverse order.

We describe the two curves by forth-order polynomials each

$$T_l(a_i, x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3 + a_5 x^4 \quad (6)$$

$$T_s(b_i, x) = b_1 + b_2 x + b_3 x^2 + b_4 x^3 + b_5 x^4 \quad (7)$$

with the parameter sets a_i and b_i.

The constraints are defined for the congruently melting composition x_c by

$$T_l'(a_i, x_c) = 0 \quad (8)$$

$$T_s'(b_i, x_c) = 0 \quad (9)$$

$$T_l(a_i, x_c) - T_s(b_i, x_c) = 0\,. \quad (10)$$

The first two constraints could be imposed separately on each of the fit curves, the third one, however, connects the two curves. Therefore, we have to combine all data to one large over-determined set of linear equations

$$\begin{bmatrix} 1 & x_{l,1} & \cdots & x_{l,1}^4 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{l,m} & \cdots & x_{l,m}^4 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & x_{s,1} & \cdots & x_{s,1}^4 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 1 & x_{s,m} & \cdots & x_{s,m}^4 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_5 \\ b_1 \\ \vdots \\ b_5 \end{bmatrix} = \begin{bmatrix} t_{l,1} \\ \vdots \\ \vdots \\ t_{l,m} \\ t_{s,1} \\ \vdots \\ \vdots \\ t_{s,m} \end{bmatrix} \tag{11}$$

which in short form may be written as

$$A \cdot p = t \tag{12}$$

with the solution for the parameters p (without constraints)

$$p = A \backslash t \; . \tag{13}$$

In MATLAB we start by constructing the matrix A and the vector t from the data sets (x_l, t_l) and (x_s, t_s)

```
A1 = makematrix(xl');
A4 = makematrix(xs');
A2 = zeros(size(A1));
A3 = zeros(size(A4));
A = [[A1,A2];[A3,A4]];
t = [tl';ts'];
```

The function `makematrix` is defined as

```
function m = makematrix(x)
m = [x.^0,x.^1,x.^2,x.^3,x.^4];
```

The first two constraints are independent from each other, thus could be introduced independently. The third one, however, has to take into account the other ones. We introduce them in stepwise chunks which are then combined to the set `Cvec` of the three constraint vectors

```
Cvec = [[0,1,2*xc,3*xc^2,4*xc^3,0,0,0,0,0];...
        [0,0,0,0,0,0,1,2*xc,3*xc^2,4*xc^3];...
```

```
        [ 1, xc, xc^2, xc^3, xc^4,...
         -1,-xc,-xc^2,-xc^3,-xc^4]];
Csta = [0;0;0];
N = size(Cvec,1);
```

After each constraint has been used it should be eliminated from the successive ones. That can be done in advance in the above set of constraints. We have to remove the upper constraints from the lower ones, i. e. have to zero the respective columns in the successive rows. As usual in such problems we define the largest element as the *pivot* element in each constraint row, normalize, and subtract iteratively in the lower rows

```
for u = 1:N,
  [V,C(u)] = max(abs(Cvec(u,:)));
  Cvec(u,:) = Cvec(u,:)/Cvec(u,C(u));
  for v = u+1:N,
    Cvec(v,:) = Cvec(v,:) - Cvec(u,:)*Cvec(v,C(u));
  end
end
```

After these preparations the problem is solved recursively

```
for n = 1:N,
    A = A - A(:,C(n))*Cvec(n,:);
    t = t - A(:,C(n))*Csta(n,:);
end;
p = A\t;
for n = N:-1:1,
    p(C(n)) = -Cvec(n,:)*p;
end;
```

In p now we have the desired parameters and can calculate the fit curves, e. g.

```
X1 = linspace(0.15,0.9,100);
Y1 = makematrix(X1')*p(1:5);
plot(X1,Y1,'k','Linewidth',3);
```

The completed phase diagram is shown in Fig. 4.

BTW, the parameters

a_1	a_2	a_3	a_4	a_5
1436.7	220.7	-293.7	152.2	-35.54
b_1	b_2	b_3	b_4	b_5
1249.8	1361.5	-2910.2	2825.4	-1063.0

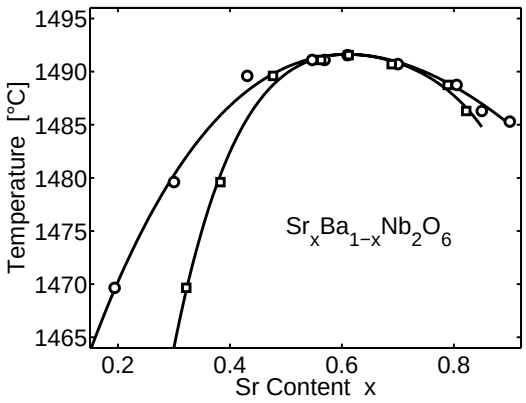

Figure 4: Phase diagram of Strontium Barium Niobate. The dots denote the measured values for the liquidus and the solidus compositions and temperatures, the black curves are fourth order polynomial fits to the data with the maximum forced to $x = 0.61$.

4 Arbitrary Functions

We discussed the constraint problem for parameter-linear fits. There the application is straight-forward and the result is always unique. If we, instead, deal with arbitrary functions, the approach is similar in mathematics but slightly different to apply. The constraints then e. g. can be introduced directly within the function called by `fminsearch`.