$$\begin{bmatrix} \sum_{i=0}^{N} x_i^0 & \cdots & \sum_{i=0}^{N} x_i^n \\ \vdots & & \vdots \\ \sum_{i=0}^{N} x_i^n & \cdots & \sum_{i=0}^{N} x_i^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{N} y_i x_i^0 \\ \vdots \\ \sum_{i=0}^{N} y_i x_i^n \end{bmatrix} \tag{3.68}$$

Insbesondere für große oder größer werdende Werte von N nehmen die Matrixelemente immer größere Werte an und das Gleichungssystem wird zusehends schlechter konditioniert. Man versucht dies zu umgehen, indem man die Elemente durch N+1 dividiert und somit mit den Mittelwerten akzeptablere Größenordnungen erzielt.

$$\begin{bmatrix} \overline{x}^0 & \cdots & \overline{x}^n \\ \vdots & & \vdots \\ \overline{x}^n & \cdots & \overline{x}^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \overline{yx}^0 \\ \vdots \\ \overline{yx}^n \end{bmatrix}$$

Führt man weiterhin die Bestimmung der Mittelwerte in rekursiver Form aus, so hat man eine sehr günstige Lösung für den Fall, dass bei Zeitfunktionen ständig neue Stützstellen bzw. Messwerte hinzukommen. Stattet man zudem noch die rekursive Mittelwertbildung mit einem nachlassenden Gedächtnis aus, so kann man sogar zeitvariante empirische Funktionen damit einigermaßen gut Approximieren, wie dies etwa in [7] dargestellt ist.

Eine Implementierung nach Gl. (3.68) könnte dann entsprechend folgender „MATLAB“-Funktion erfolgen.

```
function ret = lspoly(n, xdata, ydata, x)
% LSPOLY(n, xdata, ydata[, x])
% Least square regression with n-th order polynom for data
% set (xdata,ydata) at x
% Call: y = lspoly(2, [1:0.5:3], exp([1:0.5:3]))
%       returns the coeffs of the regression polynomial
% Call: y = lspoly(2, [1:0.5:3], exp([1:0.5:3]), 1:.1:3)
%       returns the approximated values for y=f(x)

 find coeffs by solving M * a' = g':
for k=0:n
    for j=0:n
        M(k+1,j+1)=sum( xdata.^(j+k) );
    end
end
for k=0:n
    g(k+1)=sum( ydata.*(xdata.^k) );
end
a=M\g';
cpoly=fliplr(a');
```