Neuer Algorithmus zur Regression von mehrdimensionalen Polynomen mit beliebigen Koeffizienten anhand der Methode der kleinsten Quadrate

Serge Zihlmann

ThirdWay

7. Oktober 2010

1 Abstract

Die Anpassung von Polynomen an einen gegebenen Datensatz ist eine der am häufigsten angewandten Regressionsmethoden. Einfache Probleme lassen sich leicht mit Standardprogrammen lösen. Werden allerdings benutzerspezifische Polynome benötigt, so müssen die Algorithmen dazu fest einprogrammiert werden, da die zu lösenden Gleichungssysteme schnell komplex werden. Noch schwieriger ist der Fall, wenn das gesuchte Polynom von mehreren freien Variabeln abhängen soll.

Zum Einsatz kommen in dieser Arbeit Potenzenmatrizen, welche systematisch aufgebaut werden und die Struktur der Gleichungssysteme repräsentieren. Sind diese Matrizen aufgestellt, können beliebig viele Messwerte verrechnet werden.

Dieser neu vorgestellte Algorithmus löst zwei Probleme auf einmal. Das gesuchte Polynom kann bequem über eine Potenzenmatrix eingegeben und in Sekundenschnelle angepasst werden. Die Abhängigkeit von mehreren Variabeln wird leicht bewältigt und er ist schnell[1].

[1] 1.45 Sekunden für 55'696 Messwerte in 2 Dimensionen und mit 15 Koeffizienten in Scilab

Inhaltsverzeichnis

2 Einleitung / Problemstellung

Es existiert eine Vielzahl von Algorithmen zur Berechnung von Polynomkoeffizienten zu einem Datensatz anhand der Gauss'schen Methode der kleinsten Quadrate mittels linearen Gleichungssystemen. Viele dieser Algorithmen sind schneller, die vorliegende Arbeit stellt keinen Anspruch, diese Situation zu verbessern. Ebenso sind diese Algorithmen für mehrere Dimensionen geeignet. Ein Schwachpunkt dieser Algorithmen ist allerdings, dass die gewünschten Polynome mit ihren Koeffizienten fix einprogrammiert oder nach einem Muster gebildet werden müssen, um anschliessend lösbare Gleichungssysteme aufstellen zu können. So sind diese Algorithmen beispielsweise geeignet das eindimensionale Polynom

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_n x^n \tag{2.1}$$

einem gegebenen Datensatz anzupassen. Für eine zweidimensionale Situation könnte ein Polynom wie folgt aussehen:

$$f(x;y) = a_0 + a_1 x + a_2 y + a_3 xy + a_4 x^2 y + a_5 x y^2 + a_6 x^2 y^2 ... \tag{2.2}$$

In den Algorithmen kann meist der höchste Grad angegeben werden. Die Koeffizienten werden dabei nach vorherigem Muster gebildet. Die Algorithmen stellen kompliziert und kaum überschaubar ihre Gleichungssysteme dazu auf.

Der vorliegende Algorithmus stellt eine sehr einfache Lösung dar, wenn nur *partikuläre* Koeffizienten gefragt sind. Das Aufstellen der Gleichungssysteme ist nach dem gängigen System nicht mehr möglich. Folgendes Polynom liefert ein Beispiel:

$$f(x;y) = a_0 + a_1 x + a_2 y^{27} \tag{2.3}$$

Die Koeffizienten folgen keinem Muster. Gleichungssysteme lassen sich nur schwer aufstellen.

3 Problembeschreibung und Lösung

3.1 Methode der kleinsten Quadrate

Als Beispiel sei ein anschauliches Polynom in 2 Dimensionen x, y mit 4 Koeffizienten gegeben. Es muss ein Datensatz von x, y, z Werten bestehen.

$$f(x;y) = a_0 + a_1 x + a_2 x y^2 + a_3 y^{27} \tag{3.1}$$

Der Fehler zwischen einem Messwert und dem Regressionspolynom ist gegeben als (i: i-ter Messwert):

$$r_i = f(x_i;y_i) - z_i \tag{3.2}$$

Da sich bei einer Summation der Regressionsfehler positive und negative Fehler kompensieren würden, muss das Vorzeichen eliminiert werden. Dazu wird der Fehler r quadriert. Etwas unschön dabei ist allerdings, dass starke Aussreisser in einer Messung stärker gewichtet werden.

$$e_i = r_i^2 = (f(x_i;y_i) - z_i)^2 \tag{3.3}$$

Der Gesamtfehler ergibt sich durch Summation aller Teilfehler über alle Messwerte i.

$$E = \sum_{i=1}^{n} e_i = \sum_{i=1}^{n} (f(x_i;y_i) - z_i)^2 \tag{3.4}$$

Mit dem Polynom eingesetzt folgt:

$$E = \sum_{i=1}^{n} (a_0 + a_1 x_i + a_2 x_i y_i^2 + a_3 y_i^{27} - z_i)^2 \tag{3.5}$$

3.1.1 Minimieren des Fehlers

Die Abweichung E muss nun in Abhängigkeit der zu bestimmenden Koeffizienten $a_{0..3}$ einen minimalen Wert annehmen. Daher müssen die partiellen Ableitungen von E nach

den Koeffizienten verschwinden. Es ergeben sich folgende Gleichungen:

$$\frac{\partial E}{\partial a_0} = \sum_{i=1}^{n} 2 \cdot (a_0 + a_1 x_i + a_2 x_i y_i^2 + a_3 y_i^{27} - z_i) \cdot (1) = 0 \tag{3.6}$$

$$\frac{\partial E}{\partial a_1} = \sum_{i=1}^{n} 2 \cdot (a_0 + a_1 x_i + a_2 x_i y_i^2 + a_3 y_i^{27} - z_i) \cdot (x_i) = 0 \tag{3.7}$$

$$\frac{\partial E}{\partial a_2} = \sum_{i=1}^{n} 2 \cdot (a_0 + a_1 x_i + a_2 x_i y_i^2 + a_3 y_i^{27} - z_i) \cdot (x_i y_i^2) = 0 \tag{3.8}$$

$$\frac{\partial E}{\partial a_3} = \sum_{i=1}^{n} 2 \cdot (a_0 + a_1 x_i + a_2 x_i y_i^2 + a_3 y_i^{27} - z_i) \cdot (y_i^{27}) = 0 \tag{3.9}$$

Da die Gleichungen immer Null ergeben müssen, kann der Faktor 2 zurückgelassen werden. Durch Ausmultiplizieren und versetzen des z_i Faktors auf die andere Seite folgt:

$$a_0 \sum_{i=1}^{n} 1 + a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i y_i^2 + a_3 \sum_{i=1}^{n} y_i^{27} = \sum_{i=1}^{n} z_i \tag{3.10}$$

$$a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^2 y_i^2 + a_3 \sum_{i=1}^{n} x_i y_i^{27} = \sum_{i=1}^{n} z_i x_i \tag{3.11}$$

$$a_0 \sum_{i=1}^{n} x_i y_i^2 + a_1 \sum_{i=1}^{n} x_i^2 y_i^2 + a_2 \sum_{i=1}^{n} x_i^2 y_i^4 + a_3 \sum_{i=1}^{n} x_i y_i^{29} = \sum_{i=1}^{n} z_i x_i y_i^2 \tag{3.12}$$

$$a_0 \sum_{i=1}^{n} y_i^{27} + a_1 \sum_{i=1}^{n} x_i y_i^{27} + a_2 \sum_{i=1}^{n} x_i y_i^{29} + a_3 \sum_{i=1}^{n} y_i^{54} = \sum_{i=1}^{n} z_i y_i^{27} \tag{3.13}$$

Oder in Matrizen Darstellung:

$$\sum_{i=1}^{n} \begin{pmatrix} 1 & x_i & x_i y_i^2 & y_i^{27} \\ x_i & x_i^2 & x_i^2 y_i^2 & x_i y_i^{27} \\ x_i y_i^2 & x_i^2 y_i^2 & x_i^2 y_i^4 & x_i y_i^{29} \\ y_i^{27} & x_i y_i^{27} & x_i y_i^{29} & y_i^{54} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \sum_{i=1}^{n} z_i \begin{pmatrix} 1 \\ x_i \\ x_i y_i^2 \\ y_i^{27} \end{pmatrix} \tag{3.14}$$

Ein gängiger Lösungsansatz ist nun, diese Gleichungen fest einzuprogrammieren, was es allerdings sehr erschwert, dem Anwender zur Laufzeit die Wahl von Koeffizienten offen zu lassen.

3.2 Neue Methode zur Aufstellung der Gleichungssysteme

Der Algorithmus, welcher hier vorgestellt wird, umgeht keineswegs die eben aufgestellten Gleichungssysteme. Er stellt lediglich eine einfache Möglichkeit dar, diese Gleichungen

aufzustellen. Um das Lösen des Systems mittels dem Eliminationsverfahren kümmert er sich ebenso wenig.

Die Elemente der Matrizen müssen bestimmt werden, um die Glg. anschliessend zu lösen. Dazu müssen bei jedem Element die Messwerte mit Potenzen versehen (siehe oben), multipliziert und anschliessend die Produkte für alle i summiert werden. Der Vorgang soll möglichst kompakt in Programmiercode verpackt werden.

3.2.1 Potenzenmatrizen

Die Methode macht es sich zu Nutzen, dass sich gleiche multiplizierte Faktoren in den Potenzen addieren. Der Algorithmus stellt sich im Vorhinein Tabellen mit den gemerkten Potenzen für die einzelnen Elemente der Matrix auf, um sie nachher bei der Berechnung der Einträge einfach abzurufen. Wird die Gleichung 3.14 herangezogen und in eine Matrix und einen Vektor gleicher Grösse nur die *Potenzen* für x bzw. y geschrieben, wird folgendes erhalten:

$$x: \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \tag{3.15}$$

$$y: \begin{pmatrix} 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \\ 2 & 2 & 4 & 29 \\ 27 & 27 & 29 & 54 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 2 \\ 27 \end{pmatrix} \tag{3.16}$$

Wären diese Tabellen bereits von Anfang an gegeben, so wäre das Aufstellen der Matrix ein Einfaches. Bei jedem Element müssen dann nur für alle Messwerte diese Potenzen herangezogen werden, x_i und y_i damit potenziert und beide Werte multipliziert werden. Über alle Messwerte muss dann die Summe dieses Produktes gebildet werden.

3.2.2 Summation der Potenzen

Die Frage bleibt nur, wie diese Potenzenmatrizen im Vorhinein gewonnen werden. Da für jeden Koeffizienten eine Gleichung aufgestellt werden kann, ist die Matrix immer quadratisch und eindeutig lösbar. Da die Ableitungen nach den Koeffizienten und nicht nach x oder y gebildet werden, verändern sich die Potenzen durch die Differentiation nicht. Nachstehend sind nochmals das ursprüngliche Polynom und die erste Ableitung der Fehlerquadratsumme aufgetragen:

$$f(x;y) = a_0 + a_1 x + a_2 x y^2 + a_3 y^{27} \tag{3.17}$$

$$\frac{\partial E}{\partial a_1} = \sum_{i=1}^{n} 2 \cdot (a_0 + a_1 x_i + a_2 x_i y_i^2 + a_3 y_i^{27} - z_i) \cdot (x_i) = 0 \tag{3.18}$$

Leicht zu erkennen ist, dass die Ableitungen immer gleiches Schema besitzen. Summenzeichen, Faktor 2, die ursprüngliche Gleichung minus den z-Wert und einen Faktor der nun abhängig von dem Koeffizienten ist, nach dem abgeleitet wird. Es ist deshalb erkennbar, dass die Potenzen immer die Grundform der Polynom-Gleichung besitzen. Hinzu kommt die jeweilige Potenz des abgeleiteten Koeffizienten. Die Potenzen in x und y des Polynoms werden nun in einen Vektor geschrieben:

$$x : \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} ; x : \begin{pmatrix} 0 \\ 0 \\ 2 \\ 27 \end{pmatrix} \tag{3.19}$$

Da das Polynom immer die Grundform darstellt, wird eine quadratische Matrix mit eben erhaltenen x und y Potenzen gefüllt.

$$x : \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} ; y : \begin{pmatrix} 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \end{pmatrix} \tag{3.20}$$

Durch die Ableitung wird die i-te Zeile in der Matrix mit dem zugehörigen x, y Paar des i-ten Koeffizienten multipliziert. Die Zahlen (also die Potenzen) der vorgestellten Matrizen und Vektoren dürfen deshalb einfach elementweise addiert werden. Es folgt:

$$x : \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} +^{1} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \tag{3.21}$$

$$y : \begin{pmatrix} 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \end{pmatrix} +^{2} \begin{pmatrix} 0 \\ 0 \\ 2 \\ 27 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \\ 2 & 2 & 4 & 29 \\ 27 & 27 & 29 & 54 \end{pmatrix} \tag{3.22}$$

Da die rechte Seite des Gleichungssystems 3.14 nur aus dem z-Faktor bestand (vor Ableiten) und kein x und y darin vorkommt ergibt sich dieser Vektor direkt aus den Potenzen des Polynoms. Ist also gleich

$$x : \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} y : \begin{pmatrix} 0 \\ 0 \\ 2 \\ 27 \end{pmatrix} \tag{3.23}$$

in x und y. Diese Potenzenmatrizen lassen sich folglich äusserst leicht aufstellen.

[1] Elementweise
[2] Elementweise

3.3 Anwendung

An dieser Stelle soll nochmals konkret aufgezeigt werden, wie der Algorithmus aufgebaut werden muss.

Einem Programmiercode müssen die x-, y-, z-Messwerte übergeben werden. Ebenso muss das Polynom dem Code zur Verfügung gestellt werden. Am besten geschieht das, indem in einem zweizeiligen Array nur die x und y Potenzen übergeben werden. Für das gegebene Polynom

$$f(x;y) = a_0 + a_1 x + a_2 x y^2 + a_3 y^{27} \tag{3.24}$$

folglich die Polynom repräsentierende Potenzenmatrix

$$\begin{pmatrix} \text{x-Potenzen} \\ \text{y-Potenzen} \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 2 & 27 \end{pmatrix} \tag{3.25}$$

Nun werden die Potenzenmatrizen aufgestellt für x und y, indem die erste bzw. zweite Zeile aus obiger Matrix extrahiert wird und so oft aneinander gereiht wird, wie Koeffizienten bestehen (quadratische Matrizen). Es folgen die noch unfertigen Potenzenmatrizen:

$$x: \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}; y: \begin{pmatrix} 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \\ 0 & 0 & 2 & 27 \end{pmatrix} \tag{3.26}$$

Die erste Zeile der erhaltenen Matrizen wird zu einem Vektor extrahiert und dann elementweise zur Matrix addiert, analog 3.22 und vorhergehende. Die Potenzen für den Lösungsvektor der Gleichung 3.14 (rechte Seite) sind die eben extrahierte Vektoren.

$$x: \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, y: \begin{pmatrix} 0 \\ 0 \\ 2 \\ 27 \end{pmatrix} \tag{3.27}$$

Um die Matrix schliesslich nach Gleichung 3.14 aufzustellen, muss von Element zu Element überall die Summe mit den Messwerten gebildet werden, indem die Potenzmatrizen herangezogen werden und der jeweilige x und y Wert potenziert wird. Auf der rechten Seite (Vektor) muss zusätzlich mit dem zugehörigen z_i-Wert multipliziert werden. Ist die Matrizengleichung aufgestellt, muss mit dem Gauss'schen Eliminationsverfahren gelöst werden.

3.4 Verallgemeinerung

3.4.1 Erweiterung auf mehrere Dimensionen

Oft hängt eine Funktion von mehr als einer Variabel ab. Es besteht eine Funktion in Abhängigkeit von n Variabeln (n Dimensionen).

$$f(D_1, D_2, ..., D_n) \tag{3.28}$$

Das erläuterte Prinzip mit den Potenzenmatrizen kann nun ganz einfach auf beliebig viele Dimensionen erweitert werden. Für jede neue Dimension ergibt sich demnach eine neue Potenzenmatrix, sowie einen Potenzenvektor. Ebenso muss der Polynom repräsentierenden Matrix (siehe 3.25) eine neue Zeile mit den Potenzen der neuen Dimension hinzugefügt werden.

Ein anschauliches Beispiel bietet sich für die Berechnung des spezifischen elektrischen Widerstandes eines kreisrunden Leiters. Der Widerstand ist gegeben als:

$$R(l,d) = \rho\frac{l}{A} = \rho\frac{4\cdot l}{d^2\pi} = \underbrace{4\frac{\rho}{\pi}}_{K} l \cdot d^{-2} \tag{3.29}$$

Gemessen wird nun der Widerstand für verschiedene Längen und Dicken. Die Faktoren 4, π und ρ werden zusammen zu einer Konstanten K gefasst, welche sich nun durch die Regression bestimmen liesse. Das Polynom müsste im Algorithmus in folgender Form eingegeben werden:

$$\begin{pmatrix} 1 \\ -2 \end{pmatrix} \tag{3.30}$$

Das Polynom hängt nun von zwei freien Variabeln ab, jedoch ist nur ein Summand vorhanden (ein Koeffizient zu bestimmen).

3.4.2 Negative Potenzen

Es können auch leicht gebrochene Polynome dem Algorithmus zugeführt werden. Bsp.:

$$f(x,y) = a_0 + a_1 x + a_2\frac{y}{x^2} + a_3 x^2 y + a_4\frac{1}{y^4} \tag{3.31}$$

Die Polynom repräsentierende Potenzenmatrix besitzt nun folgende Form:

$$\begin{pmatrix} \text{x-Potenzen} \\ \text{y-Potenzen} \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 & -2 & 2 & 0 \\ 0 & 0 & 1 & 1 & -4 \end{pmatrix} \tag{3.32}$$

Achtung: Wenn negative Potenzen verwendet weden, dann dürfen sich keine Datenpunkte in der Messreihe befinden, die in einer Dimension Null werden, da ansonsten Divisionen durch Null entstehen.

3.4.3 Reelle Potenzen

Dem Algorithmus können, wie im letzten Abschnitt beschrieben, beliebige Potenzen zugeführt werden. Dies gilt auch für solche die *nicht* ganzzahlig sind. Bsp.:

$$f(x,y) = a_0 + a_1 x + a_2 \frac{y^{0.5}}{x^2} + a_3 x^{2.79} y^{-0.9} + a_4 \frac{1}{y^{4.1}} \tag{3.33}$$

Die Polynom repräsentierende Potenzenmatrix besitzt nun folgende Form:

$$\begin{pmatrix} 0 & 1 & -2 & 2.79 & 0 \\ 0 & 0 & 0.5 & -0.9 & -4.1 \end{pmatrix} \tag{3.34}$$

3.5 Beschränkung auf lineare Systeme

Die zu bestimmenden Koeffizienten dürfen in *ausschliesslich linearer* Form vorhanden sein. Folgende Beispielsummanden sind nicht möglich:

$$f(x) = (a_0 x)^2 + sin(a_1 x) + a_2 x + a_2 x^2 \tag{3.35}$$

Bei letzten beiden Summanden wurde zweimal derselbe Koeffizient verwendet.

3.6 Rechenzeit

Die Rechenzeit nimmt quadratisch mit der Anzahl an gesuchten Koeffizienten zu, da sich aus den Gleichungen quadratische Matrizen ergeben. Mit steigender Anzahl Dimensionen nimmt der Aufwand allerdings nur linear zu. Der Algorithmus ist trotzdem schnell. In Scilab benötigte der Algorithmus[3] lediglich 1.45 Sekunden um 55'696 Messwerte in 2 Dimensionen mit 15 Koeffizienten zu verarbeiten.

[3] Siehe Algorithmenvorschlag am Ende

4 Algorithmenvorschlag

4.1 Primärer Scilab Algorithmus

Das frei erhältliche Programm Scilab verfügt über umfangreiche Funktionen. Es wurde dabei speziell Wert auf Matrizenoperationen gelegt. Der Algorithmus lässt sich deshalb in sehr kompakter Form darstellen.

Dem Algorithmus müssen drei Variabeln übergeben werden; die unabhängigen Messwertpositionen „data“, die zugehörigen Messwerte „zval“ sowie das Polynom. Der Aufruf der Funktion erfolgt mit „PowerFit(data, zval, polynome)“. Die Daten müssen folgende Form besitzen:

$$data = \overset{\text{Dimensionen}\rightarrow}{\begin{pmatrix} D_{1,1} & D_{2,1} & \dots & D_{d,1} \\ D_{1,2} & D_{2,2} & \dots & D_{d,2} \\ \vdots & \ddots & & \\ D_{1,n} & D_{2,n} & \dots & D_{d,n} \end{pmatrix}}, \; zval = \begin{pmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_n \end{pmatrix}$$

n steht für die Anzahl Messwerte. d sind die Anzahl Dimensionen, in der das Polynom definiert ist. Das Polynom muss in folgender Form eingegeben werden:

$$polynome = \begin{pmatrix} P(D_1,1) & P(D_1,2) & \cdots & P(D_1,k) \\ P(D_2,1) & P(D_2,2) & \cdots & P(D_2,k) \\ \vdots & \ddots & & \\ P(D_d,1) & P(D_d,2) & \cdots & P(D_d,k) \end{pmatrix}$$

Wobei der Index D_i beschreibt in welcher Dimension die eingetragene Potenz P wirkt. k beschreibt die gesamte Anzahl an Koeffizienten des Polynoms, d beschreibt die Anzahl Dimensionen.

```
1 //-----------------------------------------------------------------------
2 // This algorithm calculates regressions polynomes of various
3 // dimensions, coefficients and any combination of potences.
4 // Released 22. September 2010 by Serge Zihlmann
5 // THIS CODE IS FREE TO BE USED IN ANY KIND OF SOFTWARE
6 // For further questions contact: powerfit( at@at )thirdway.ch
7 //-----------------------------------------------------------------------
8 //-----------------------------------------------------------
```

```
9  function [y] = PowerFit(data, zval, polynome)
10
11   [ndata dim] = size(data);          // Get amount of datapoints
12
13   // Get number of free variables of polynome
14   [dim nCoeff] = size(polynome);  //Number of dimensions and coefficients
15
16   // Generate exponent tables
17   ExpMat = zeros(nCoeff,nCoeff,dim);
18   for i=1:nCoeff
19     for k=1:dim
20       ExpMat(i,:,k) = polynome(k,:)+polynome(k,i);
21     end
22   end
23   ExpVec = polynome';
24
25   // Generate matrices to fill
26   A = zeros(nCoeff,nCoeff); // Final matrix
27   b = zeros(nCoeff,1);        // Final solution vector
28
29   // Build up matrix A
30   for i=1:nCoeff
31     for k=1:nCoeff
32       t = ones(ndata,1);          // Temporary vector
33       for r=1:dim
34         t = t .* (data(:,r).^ExpMat(i,k,r));
35       end
36       A(i,k) = sum(t);
37     end
38   end
39
40   // Build up vector b
41   for i=1:nCoeff
42       t = zval;
43       for k=1:dim
44         t = t .* (data(:,k).^ExpVec(i,k));
45       end
46       b(i) = sum(t);
47   end
48
49   // Solve the system A*y=b for y
50   y = inv(A)*b; // Calculate solution, return parameters
51 endfunction
52 //-----------------------------------------------------------------
```

In Zeile (11) wird der Umfang des Datensatzes abgefragt. (14) bestimmt die Anzahl Koeffizienten und Dimensionen des eingegebenen Polynomes. Die Dimension des Datensatzes muss zwingend mit dem des definierten Polynomes übereinstimmen.

Im nächsten Schritt (17-23) werden die in den vorhergehenden Kapiteln beschriebenen Potenzen Vektoren bzw. Matrizen generiert. Das Gleichungssystem $A \cdot y = b$ wird in

den Zeilen (30-47) aufgebaut. Dazu wird die Matrix A Elementweise gefüllt, wobei die Potenzenmatrix herangezogen wird. Der Lösungsvektor b wird nach gleichem Schema mittels dem Potenzenvektor gebildet.

Zeile (50) löst das resultierende Gleichungssystem und liefert die Polynomkoeffizienten y.

4.2 Kompletter Funktionssatz

Im letzten Abschnitt wurde nur der primäre Algorithmus besprochen. Zwei weitere Funktionen sind allerdings sehr hilfreich. „PrettyPrintF“ gibt das definierte Polynom in einer schön lesbaren Form wieder. Der Funktion kann neben dem Polynom auch eine neue Beschriftung für die Potenzen und Koeffizienten gegeben werden. Siehe Beispiel 1 & 2. Die zweite Funktion „EvalPol“ lässt das Polynom mit den bereits bestimmten Koeffizienten an einem Punkt auswerten.

```
1  //-------------------------------------------------------------------------
2  // This algorithm calculates regressions polynomes of various
3  // dimensions, coefficients and any combination of potences.
4  // Released 22. September 2010 by Serge Zihlmann
5  // THIS CODE IS FREE TO BE USED IN ANY KIND OF SOFTWARE
6  // For further questions contact: powerfit( at@at )thirdway.ch
7  //-------------------------------------------------------------------------
8  //-----------------------------------------------------------
9  function [y] = PowerFit(data, zval, polynome)
10
11   Error = 0;                  //Start without errors
12   [ndata dimD] = size(data); //Get amount of datapoints
13
14   //get number of free variables of polynome
15   [dim nCoeff] = size(polynome); //Number of dimensions and coefficients
16
17   //Check for Errors
18   if (dim <> dimD)
19     printf("Different dimensions of polynome and data\n");
20     Error = 1
21   end
22    if (ndata <> size(zval))
23      printf("Different amount of data\n");
24     Error = 1
25   end
26
27
28   if (Error <> 1) //only start if no errors occur
29   //generate exponent tables
30   ExpMat = zeros(nCoeff,nCoeff,dim);
31   for i=1:nCoeff
32     for k=1:dim
```

```
33        ExpMat(i,:,k) = polynome(k,:)+polynome(k,i);
34      end
35    end
36    ExpVec = polynome';
37
38    // Generate matrices to fill
39    A = zeros(nCoeff,nCoeff); // Final matrix
40    b = zeros(nCoeff,1);      // Final solution vector
41
42    // Build up matrix A
43    for i=1:nCoeff
44      for k=1:nCoeff
45        t = ones(ndata,1);          // Temporary vector
46        for r=1:dim
47          t = t .* (data(:,r).^ExpMat(i,k,r));
48        end
49        A(i,k) = sum(t);
50      end
51    end
52
53    // Build up vector b
54    for i=1:nCoeff
55        t = zval;
56        for k=1:dim
57          t = t .* (data(:,k).^ExpVec(i,k));
58        end
59        b(i) = sum(t);
60    end
61
62    // Solve the system A*y=b for y
63    y = inv(A)*b; //calculate solution, return parameters
64    end // end error
65  endfunction
66  //-----------------------------------------------------------------
67  //Evaluates polynome with coefficients, x=datapoint
68  function [y]=EvalPol(polynome, x, coefficients)
69    [cx cy] = size(polynome);
70    y = coefficients';
71    for i=1:cx
72      y = y.*(x(i).^polynome(i,:));
73    end
74    y = sum(y);
75  endfunction
76  //-----------------------------------------------------------------
77  //--Prints the input polynome in a pretty shape
78  function [y]=PrettyPrintF(polynom, coeffName, dimName)
79    //prints out the symbolic function defined in polynom
80    [cx cy] = size(polynom)
81    printf("Regression function:\n");
82    printf("---------------------\nf(");
83
84    [ax ay] = size(dimName);
```

```
85    [bx by] = size(coeffName);
86
87    for i=1:cx
88      if (i<=ay)
89        printf("%s", dimName(i));
90      else
91        printf("D%i",i);
92      end
93      if(i<>cx) printf(","); end
94    end
95    printf(")␣=␣");
96    U = [];
97    for i=1:cy
98      if (i<=by)
99        U = U + coeffName(i);
100     else
101       U = U + "a"+string(i);
102     end
103
104     for k=1:cx
105       if(polynom(k,i)<>0)//if power of x is greater than zero
106         if (k<=ay)
107           U = U + dimName(k);
108         else
109           U = U + "D" + string(k);
110         end
111
112         if((polynom(k,i)<>0) & (polynom(k,i)<>1))
113           U = U+"^"+string(polynom(k,i));
114         end
115       end
116     end
117
118     if(i<cy)
119       U = U + "␣+␣";
120     end
121   end
122   printf("%s",U);
123   printf("\n\n");
124   y = []; //dont return anything
125 endfunction
```

4.2.1 EvalPol()

Die Funktion wird EvalPol(A,B,C) aufgerufen. A ist das Polynom in Potenzenform. B ist die Stelle an der das Pol. ausgewertet werden soll und C sind die berechneten Koeffizienten.

4.2.2 PrettyPrintF()

Die Funktion wird PrettyPrintF(A,B,C) aufgerufen. A ist wiederum das Pol. in Potenzenform. B & C sind char arrays, welche die Benennung der unabhängigen Variabeln, sowie den Koeffizienten ermöglichen. Die Funktion kann auch PrettyPrintF(A,[ ],[ ]) aufgerufen werden, also ohne benennende Matrizen. Die Benennung wird dann automatisch generiert.

4.3 Beispiele zum Aufruf

4.3.1 Beispiel 1

```
1  //-------------------------------------------------------------------
2  // This is a simple example with only basic use. Labeling
3  // for coefficients and dimensions is automatic. No values
4  // will be printed out
5  //-------------------------------------------------------------------
6
7  // Restart programme
8  clear;     // Clear memory
9  clc;       // Clear display
10 exec('Functions.sci'); // Get functions from extern file
11
12 // Enter data position
13 data =[
14 2 2
15 2 3
16 2 4
17 2 5
18 3 2
19 3 3
20 3 4
21 3 5
22 ];
23
24 // Enter values
25 zval =[
26 45
27 63
28 83
29 103
30 93
31 138
32 183
33 228
34 ];
35
```

```
36 // Input the polynome
37 polynome =[
38 0 1 2
39 0 1 1
40 ];
41
42 PrettyPrintF(polynome, [], []);     // Print polynome in pretty shape
43 y = PowerFit(data,zval,polynome);   // Calc polynome (main function)
44
45 x=[2,2];
46
47 disp(EvalPol(polynome,x,y))
48
49 clf;
50
51 xx=0:0.1:6;;
52 yy=-6:0.1:7;
53
54 for i=1:length(xx)
55   for k=1:length(yy)
56     zz(i,k) = EvalPol(polynome,[xx(i),yy(k)],y)
57   end
58 end
59
60
61 plot3d1 (data(:,1), data(:,2), (zval),flag=[-1, 2, 3]);
62 plot3d1 (xx, yy, (zz),flag=[-1, 2, 3]);
```

4.3.2 Beispiel 2

```
1  //-----------------------------------------------------------
2  // This is an asvanced example which also defines your own
3  // labeling for coefficients and dimensions. Furthermore
4  // plots are generated and values wiht the calculated poly-
5  // nome are ploted. But the data is only one dimensional
6  //-----------------------------------------------------------
7
8  // Restart programme
9  clear;     // Clear memory
10 clc;       // Clear display
11 clf;       // Clear plot
12 exec('Functions.sci'); //Get functions from extern file
13
14 // Enter measured data: Main input section
15 data =[      // Define dataset
16 1;2;3;3.5;4;5;6];
17
18 zval =[      // Define values according to dataset
19 9;2;3;4.5;6;5;4];
20
```

```
21 // Input the polynome
22 polynome =[
23 0 1 -1 2
24 ];
25
26 // Enter names for coefficients and dimensions
27 dimNames   = ["x"];    // Label of dimensions
28 coeffNames = ["a" "b" "c" "d" "e" "f" "g"];    // Names of dimensions
29
30 // Generate automatic names if missing
31 [cx cy] = size(coeffNames);
32 for i=1:length(polynome(1,:))
33   if (i>cy)
34     coeffNames(1,i) = "a" + string(i);
35   end
36 end
37 clear cx, cy;
38
39 // Printout polynome formula in pretty shape
40 PrettyPrintF(polynome, [], []);                  // Print with auto names
41 PrettyPrintF(polynome, coeffNames, dimNames);  // Print with own names
42
43 // Printout results (main calculation)
44 printf("Result:\n--------\n");
45 y = PowerFit(data,zval,polynome);  // Calc polynome (main function)
46 [cx cy] = size(y);
47 disp([coeffNames(1:cx)' string(y)]);      // Printout coefficients
48 clear cx, cy;
49
50 // Generate nice plots
51 plot(data,zval,"or");                  // Plot entered data
52 D = (max(data)-min(data))*1.05/2;
53 C = (min(data)+max(data))/2;
54 xx = C-D:0.1:C+D;                      // Generate x area
55 for i=1:length(xx)                     // Calculate corresponding values
56   yy(i) = sum((xx(i).^polynome).*(y'));
57 end
58 plot(xx,yy)                            // Plot regression polynome
```

Die Scilab-Files mit zusätzlichen Beispielen können im Übrigen unter http://www.thirdway.ch herunter geladen werden.
