Kapitel 14

Anwendungen

14.1 Kurvenanpassung - Fitten

Kurvenanpassung, oder auch Fitten genannt, ist eine Technik mit der man versucht, eine gegebene mathematische Modellfunktion bestmöglich an Datenpunkte anzupassen. Der einfachste Fall ist wohl die Bestimmung einer Ausgleichsgeraden, wo die Koeffizienten k und d des Polynoms ersten Grades $f(x, k, d) = kx + d$ so bestimmt werden, dass die Summe der Abstandsquadrate von den Datenpunkten den kleinstmöglichen Wert annimmt.

Diese Minimierung der Summe der Abstandsquadrate bezeichnet man als "Least Squares"-Verfahren. Bei Vorliegen von m Datenpunkten (x_j, x_j) und unter der Annahme, dass die Modellfunktion mit $f(x, a)$ bezeichnet wird, kann die Summe der Abstandsquadrate geschrieben werden als

$$q = \sum_{j=1}^{m} (y_j - f(x_j, a))^2 \stackrel{!}{=} \text{Min} . \tag{14.1}$$

Diese Summe q der Abstandsquadrate soll minimiert werden, d.h. man sucht nach den besten Parametern a der Funktion $f(x, a)$, wobei a am Beispiel der Ausgleichsgeraden der Vektor $[k, d]$ ist.

Im Wesentlichen sind zwei unterschiedliche Klassen von Modellfunktionen zu unterscheiden, solche die lineare Funktionen der Parameter a_i sind und solche die nichtlineare Funktionen der Parameter a_i sind. Lineare Funktionen sind Polynome bzw. Funktionen, die man im weitesten Sinne als verallgemeinerte Polynome bezeichnen

könnte

$$f(x,a) = \sum_{i=0}^{n} a_i x^i \ , \tag{14.2}$$

$$f(x,a) = a_0 + a_1 x + a_2 x^2 \ , \tag{14.3}$$

$$f(x,a) = \sum_{i=0}^{n} a_i f_i(x) \ , \tag{14.4}$$

$$f(x,a) = a_0 + a_1 \sin x + a_2 \cos x \ . \tag{14.5}$$

Manche Modellfunktionen können z.B. durch Logarithmierung linearisiert werden

$$f(x,a) = a_0 \exp\left(-a_1 x\right) \qquad \Rightarrow \hat{f}(x,a) = \ln a_0 - a_1 x \ , \tag{14.6}$$

$$f(x,a) = a_0 x^{a_1} \qquad \Rightarrow \hat{f}(x,a) = \ln a_0 + a_1 \ln x \ . \tag{14.7}$$

Typische nichtlineare Funktionen sind solche, wo die Parameter z.B. als Argument von trigonometrischen Funktionen (Frequenz, Phase) vorkommen oder andere Funktionen mit komplexeren funktionalen Zusammenhang

$$f(x,a) = a_0 \sin\left(a_1 x + a_2\right) \ , \tag{14.8}$$

$$f(x,a) = a_0 \exp\left(\frac{-(x-a_1)^2}{a_2}\right) \ , \tag{14.9}$$

$$f(x,a) = \frac{a_0}{a_1 + a_2 x} \ . \tag{14.10}$$

14.1.1 Auswahl der Modellfunktion

Ein wichtiger Schritt bei der Kurvenanpassung ist die Auswahl der Modellfunktion. Wann immer es möglich ist, lässt man sich von einem zugrunde liegenden physikalischen Modell bei der Auswahl leiten:

- Schwingung - Kombination trigonometrischer Funktionen
- Dämpfung, Abklingverhalten - Exponentialfunktion
- Theoretisches Modell

Ist die Wahl auf ein lineares Modell in Bezug auf die Parameter gefallen, kann man den Anweisungen in Abschnitt 14.1.2 folgen, sonst den Anweisungen im Abschnitt 14.1.4.

14.1.2 Lineares Fitten

Um lineares Fitten zu verstehen, sollte man sich vergegenwärtigen, wie man ein Polynom exakt durch m-Datenpunkte $(x_j, y_j$ legen kann. Man benötigt dazu im Normalfall ein Polynom $(m-1)$-ten Grades mit m Koeffizienten. Die Koeffizienten müssen dabei folgendes bestimmtes Gleichungssystem erfüllen

$$\sum_{i=0}^{m-1} a_i x_j^i = y_j \ , \ j = 1 \dots m \ , \tag{14.11}$$

welches in Matrixform als

$$X_{ji} a_i = y_j \tag{14.12}$$

geschrieben werden kann. In den m Spalten der Matrix $X_{ji} = x_j^i$ stehen somit die Spaltenvektoren x^{m-1}, x^{m-2} $\dots$ x^0. Dieses bestimmte Gleichungssystem kann nun nach den Regeln der linearen Algebra (siehe Abschnitt Kapitel 6) gelöst werden. In MATLAB lautet die Lösung mit Hilfe der Matrix-Links-Division `a=X\y`, wobei y ein Spaltenvektor sein muss.

Im Falle des Fittens hat man es normalerweise mit einer größeren Anzahl von Datenpunkten zu tun und möchte in den seltensten Fällen Modellfunktionen mit der gleichen Anzahl von Parametern verwenden. Damit hat man es mit einem überbestimmten Gleichungssystem zu tun, dass nicht mehr exakt gelöst werden kann. In MATLAB kann man für die Lösung eines solchen überbestimmten Gleichungssystems aber die gleiche Syntax verwenden. Sobald ein Gleichungssystem überbestimmt ist, löst MATLAB bei Verwendung des Befehls `a=X\y` das Gleichungssystem im Sinne des "Least-Squares"-Verfahrens entsprechend der Gleichung 14.1.

Die Unterscheidung zwischen linearen und nichtlinearen Modellfunktionen ist deswegen so wichtig, weil in der numerischen Umsetzung wesentliche Unterschiede bestehen. Im Fall von linearen Funktionen kann das Minimierungsproblem in Gleichung 14.1 immer in ein exakt lösbares lineares Gleichungssystem überführen. Wie man sich leicht überzeugen kann, erhält man für $k = 0 \dots n$ aus $\frac{\partial q}{\partial a_k} = 0$ das lineare Gleichungssystem

$$X_{ki} a_i = b_k \ , \tag{14.13}$$

$$X_{ki} = \sum_{j=1}^{m} f_i(x_j) f_k(x_j) \ , \tag{14.14}$$

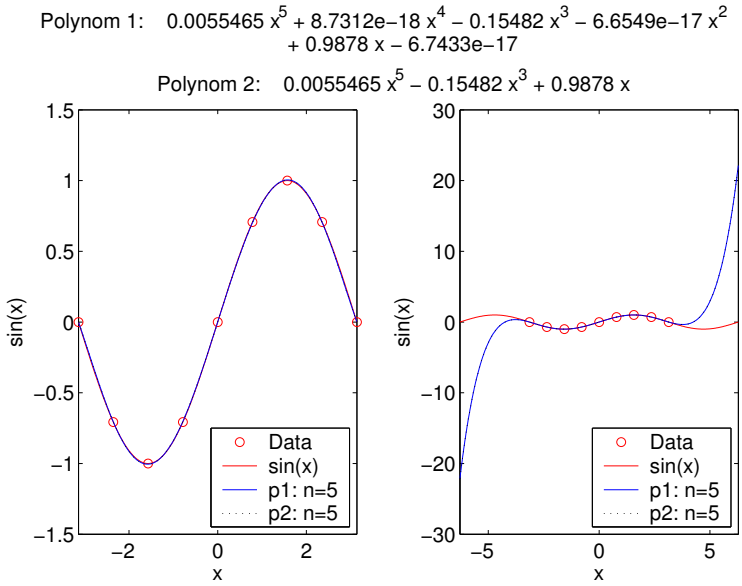
$$b_k = \sum_{j=1}^{m} y_j f_k(x_j) \ . \tag{14.15}$$

Diesen Vorteil hat man bei einem nichtlinearen Zusammenhang natürlich nicht. In diesem Fall ist man dann auf näherungsweise Verfahren zur Minimierung von Gleichung 14.1 angewiesen.

14.1.2.1 Polynom-Fit

Hat man sich für ein Polynom vom Grad n entschieden, kann man die MATLAB-Funktion `polyfit` verwenden. Die Verwendung soll hier am Beispiel eines Polynomfittes der Sinusfunktion gezeigt werden.

```
xd = linspace(-pi,pi,9); yd = sin(xd); grad = 5;
p1 = polyfit(xd,yd,grad);
x  = linspace(-pi,pi,500); y1 = polyval(p1,x);
subplot(1,2,1);plot(xd,yd,'ro',x,sin(x),'r-',x,y1,'b-');
x  = linspace(-2*pi,2*pi,500); y1 = polyval(p1,x);
subplot(1,2,2);plot(xd,yd,'ro',x,sin(x),'r-',x,y1,'b-');
```



Diese Darstellung demonstriert drei interessante Aspekte:

- Der Fit mit einem Polynom 5-ten Grades ist innerhalb des Datenbereichs sehr gut.
- Außerhalb des Datenbereichs bricht die gute Übereinstimmung sehr rasch zusammen, da Polynome natürlich nicht die beste Modellfunktion für Schwingungen sind. Der Grund dafür ist, dass alle Polynome für $x \to \pm\infty$ nach $\pm\infty$ "explodieren".
- Wie auf Grund der Reihenentwicklung für den Sinus nicht anders zu erwarten, sind die Koeffizienten für gerade Potenzen von x nahezu Null.

Will man an diesem Beispiel die Koeffizienten für die geraden Potenzen von vorne herein auf Null setzen, kann man `polyfit` nicht verwenden und muss sich auf das Lösen von überbestimmten Gleichungssystemen besinnen.

```
X = [xd(:).^5,xd(:).^3,xd(:)];
b = yd(:);
a = X \ b;
p2 = zeros(1,grad+1); p2(1:2:end) = a;
```

Hier werden nach Lösen des Gleichungssystems mit `a = X\b` die drei erhaltenen Koeffizienten an den richtigen Stellen im Polynom `p2` gespeichert.

14.1.2.2 Allgemeiner linearer Fit

Im allgemeinen Fall muss es sich nun nicht um Polynome handeln. Im folgenden Beispiel soll die Funktion $f(x, a)$, die als

$$f(x, a) = a_1 f_1(x) + a_2 f_2(x) \,, \tag{14.16}$$
$$f_1(x) = \exp\left(-0.2\ x\right) \,, \tag{14.17}$$
$$f_2(x) = \sin(4x) \exp\left(-0.4\ x\right) \,, \tag{14.18}$$

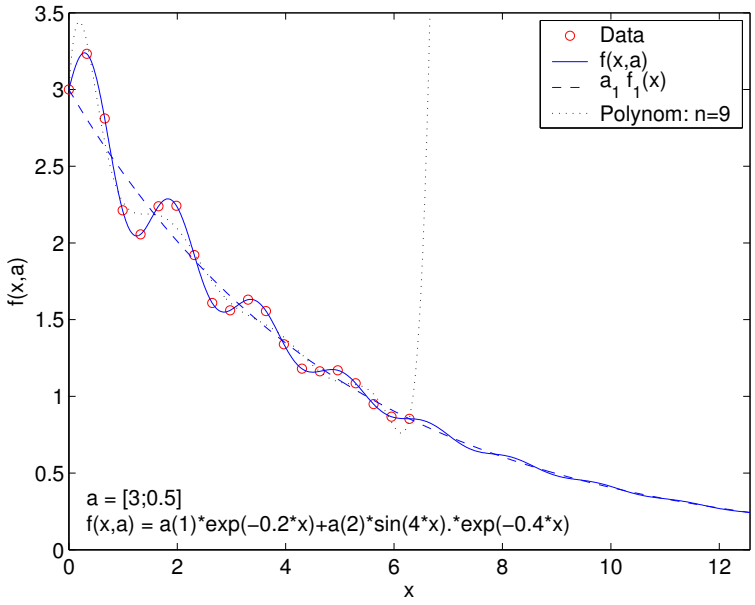
gegeben ist, an die Datenpunkte in den Vektoren `xd` und `yd` angepasst werden. Zuerst kann man in diesem Fall die Funktionen als `inline`-Funktionen definieren:

```
f1c = 'exp(-0.2*x)';
f2c = 'sin(4*x).*exp(-0.4*x)';
fc  = ['a(1)*',f1c,'+a(2)*',f2c];
f1  = inline(f1c,'x');
f2  = inline(f2c,'x');
f   = inline(fc,'x','a');
```

Um das lineare Gleichungssystem nun lösen zu können, muss man die Koeffizientenmatrix `X` und den Inhomegenitätsvektor `b` definieren und dann die Lösung für die Koeffizienten `a` bestimmen:

```
X  = [f1(xd(:)),f2(xd(:))];
b  = yd(:);
a  = X\b;
```

Mit diesen Daten kann man nun die Funktionen darstellen.



Hier fällt nun auf, dass

- bei passender Modellfunktion die Kurve auch außerhalb des Datenbereichs sich "vernünftig" verhält, und dass
- der zusätzlich eingezeichnete Polynomfit innerhalb der Daten nicht sehr gut liegt, und dass
- er natürlich außerhalb der Daten das gewohnte Verhalten zeigt.

14.1.3 Exponentieller Fit

Beim radioaktiven Zerfall folgt die Intensität der Strahlung folgendem Gesetz

$$I(t) = I_0 \exp\left(-\frac{t}{\tau}\right) \ , \tag{14.19}$$

wobei I_0 die Anfangsintensität und τ die Halbwertszeit ist. Logarithmiert man die Gleichung, kommt man zu folgender Darstellung

$$\hat{I}(t) = a_1 t + a_2 \ , \tag{14.20}$$

$$\hat{I} = \ln I \ , \tag{14.21}$$

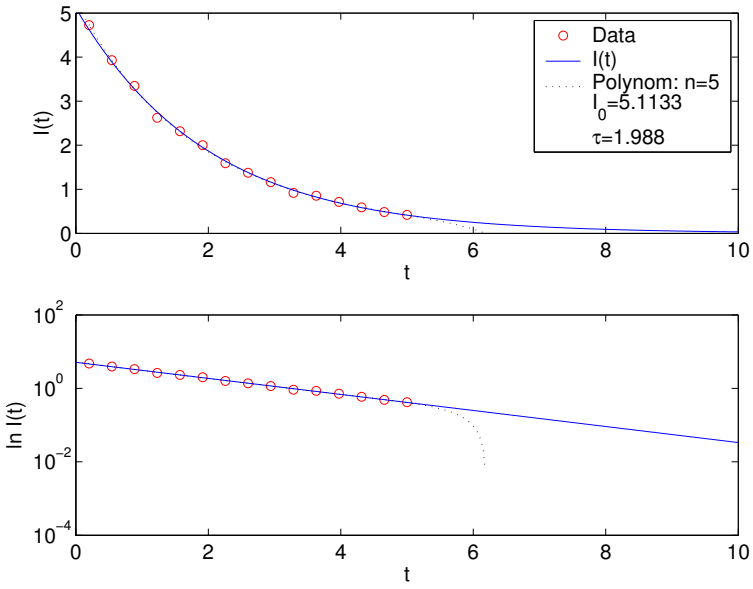
$$a_1 = -\frac{1}{\tau} \ , \tag{14.22}$$

$$a_2 = \ln I_0 \ , \tag{14.23}$$

Vorausgesetzt die Zeit- und die Intensitätswerte sind in den Variablen `t` und `I` gespeichert, kann man nun wieder das Gleichungssystem aufbauen und lösen. Hier sieht man auch, dass wenn man ein konstantes Glied bestimmen will, die Matrix `X` einfach eine Reihe mit Einsen enthalten muss:

```
X = [t(:),ones(size(t(:)))];
b = log(I(:));
a = X\b;
tau = -1 / a(1);
I0  = exp(a(2));
```

Nach Durchführen des Fits (Lösen des Gleichungssystems) kann man natürlich dann wieder die interessierenden Größen τ und I_0 berechnen.



Der zusätzlich berechnete Polynomfit erweist sich natürlich wieder als untauglich.

14.1.4 Nichtlineares Fitten

Im Falle einer Modellfunktion, die eine nichtlineare Funktion in den Modellparametern ist, muss man nun zu anderen Methoden greifen. Als einfachstes Beispiel soll hier eine Schwingung mit Amplitude A, Frequenz ω und Phase ϕ verwendet werden, die durch folgenden Zusammenhang gegeben ist

$$y(t) = A \sin(\omega t + \phi) \ . \tag{14.24}$$

Für eine Umsetzung des Problems in MATLAB muss man nun eine MATLAB-Funktion oder eine `inline`-Funktion schreiben, die den funktionalen Zusammenhänge wiedergibt:

```
fc = 'a(1)*sin(a(2)*t+a(3))';
f  = inline(fc,'a','t');
```

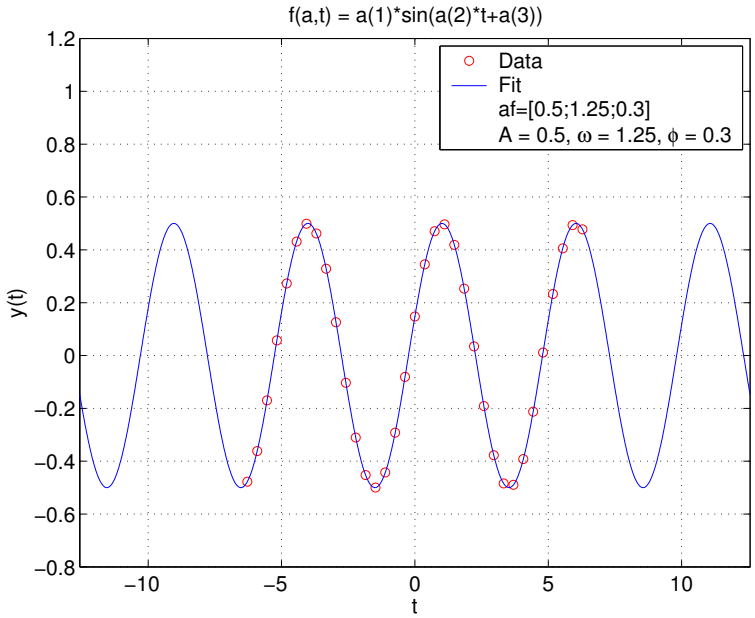
Wichtig dabei ist, dass alle Parameter (hier A, ω, ϕ) in einen Vektor zusammengefasst werden (hier `a`), und dass dieser Vektor an erster Stelle in der Übergabeliste steht.

Der Funktionsaufruf `y=f(a,t)` muss also für jeden Parametervektor `a` und jeden Zeitvektor `t` die Auslenkung `y` liefern, wobei `y` die gleiche Größe wie `t` haben muss.

In diesem Beispiel liegen die Datenpunkte wieder als Vektoren `td` und `yd` vor. Im Unterschied zum nichtlinearen Fitten braucht man nun aber auch einen Startwert für die Parameter um MATLAB mitzuteilen, wo man ungefähr die Lösung erwartet. Danach kann man mit dem MATLAB-Programm nlinfit den Fit durchführen. Diese Routine stammt aus dem MATLAB-Statistik Toolbox, ist also beim Grundpaket nicht installiert.

```
as = [0.8,1,0.2];
af = nlinfit(td,yd,f,as)
t = linspace(-4*pi,4*pi,1000);
y = f(af,t);
```

Damit kann man sich dann mit dem Zeitvektor `t` die Modellfunktion berechnen und zusammen mit den Daten darstellen.



Warum braucht man nun einen Startwert. Im Unterschied zum linearen Fitten, wo das zugehörige Gleichungssystem immer eine eindeutige Lösung besitzt, die direkt ermittelt werden kann, benötigt man hier ein iteratives Verfahren. Dabei wird ausgehend von einem Startwert das Minimum einer Funktion gesucht, in dem man

sich Schritt für Schritt dem Minimum nähert. Dazu gibt es viele Möglichkeiten (z.B.: Gauss-Newton). Entscheidend ist also der Unterschied, das es kein direktes Verfahren gibt um die Lösung zu finden. Nichtlineare Probleme haben dazu auch häufig mehrere (oft viele) "lokale" Minima, interessiert ist man aber am so genannten "globalen" Minimum, welches die "bestmögliche" Lösung des Problems darstellt. Daher ist eine gute Wahl des Startwertes meist eine essentielle Vorleistung für eine gute Lösung. Meist findet man diese durch "clevere" Betrachtung der Daten.

Als weiteres Beispiel soll hier die Funktion aus Gleichung 14.16 verallgemeinert werden, indem alle Parameter veränderlich gemacht werden,

$$f(x,a) = a_1 f_1(x) + a_2 f_2(x) \ , \tag{14.25}$$
$$f_1(x) = \exp\left(-a_3 x\right) \ , \tag{14.26}$$
$$f_2(x) = \sin(a_4 x)\exp\left(-a_5 x\right) \ , \tag{14.27}$$

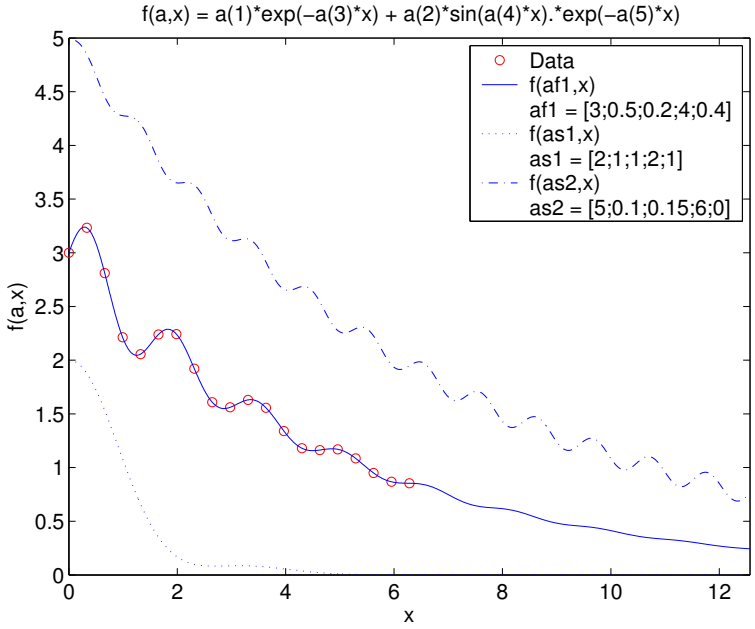
Dies ist nun ein nichtlineares Problem in a mit folgender Umsetzung in MATLAB:

```
fc  = 'a(1)*exp(-a(3)*x) + a(2)*sin(a(4)*x).*exp(-a(5)*x)';
fa  = inline(fc,'a','x');
```

Unter der Voraussetzung, dass die Daten wieder in `xd` und `yd` zur Verfügung stehen, kann man die Lösung folgendermaßen finden

```
as1 = [2,1,1,2,1];
af1 = nlinfit(xd,yd,fa,as1)
```

Die Darstellung der Daten, des Ergebnisses und der Resultate von zwei Startwerten kann man in folgender Graphik sehen:



Die Standard MATLAB-Funktion für das suchen von Minima ist die Routine `fminsearch`. Um diese beim vorliegenden Problem verwenden zu können, muss man eine Funktion programmieren, die den Skalarwert q der Gleichung 14.1 zurück gibt. Diese Funktion muss also die Summe der Abstandsquadrate an allen Datenpunkten berechnen und benötigt dafür den Parametervektor `a` und die Daten `xd` und `yd`. Am Beispiel der letzten Funktion kann das so aussehen:

```
function q = lqfunc(a,xd,yd)
y = a(1)*exp(-a(3)*xd) + a(2)*sin(a(4)*xd).*exp(-a(5)*xd);
q = sum((y-yd).^2);
```

Wichtig ist also der Punkt, dass hier jeweils nur ein skalarer Wert, nämlich der Wert der zu minimierenden Funktion, zurückgegeben wird. Diese Funktion wird nun an `fminsearch` zusammen mit Startwerten übergeben

```
af3 = fminsearch(@lqfunc,as1,[],xd,yd)
```

und liefert die "besten" Parameter. An Stelle von `[]` kann man Optionen übergeben (siehe Hilfe zu `fminsearch`). Die Datenvektoren `xd` und `yd` werden von `fminsearch` an die Funktion `lqfunc` weitergegeben. Dieses Beispiel zeigt somit nochmals, dass eigentlich die Funktion 14.1 minimiert wird, und dass dies dann zur besten Annäherung der Modellfunktion an die Daten führt.

14.2 Interpolation

Im Unterschied zur Kurvenanpassung (Fitten einer Modellfunktion) verwendet man bei Interpolieren "lokal" an die Datenpunkte angepasste Funktionen, wobei sichergestellt wird, dass diese Funktionen exakt die Datenpunkte reproduzieren. Ziel des Verfahrens ist es, zwischen den diskreten Datenpunkten (z.B. Messwerten) einen vernünftigen Verlauf zu finden (z.B. zum Plotten). In den meisten Fällen beschränkt man sich dabei auf Polynome bis maximal dritten Grades, die aber nur "lokal" um den jeweiligen Datenpunkt verwendet werden.

MATLAB bietet für eindimensionale Probleme die Routine `interp1` an, die folgenden Aufruf benötigt

```
y = interp1(xd,yd,x,method)
```

wobei in `xd` und `yd` wieder die Datenpunkte liegen, `x` ein Vektor mit meist dichter liegenden x-Werten ist und `method` eine Stringvariable mit der gewünschten Methode ist.

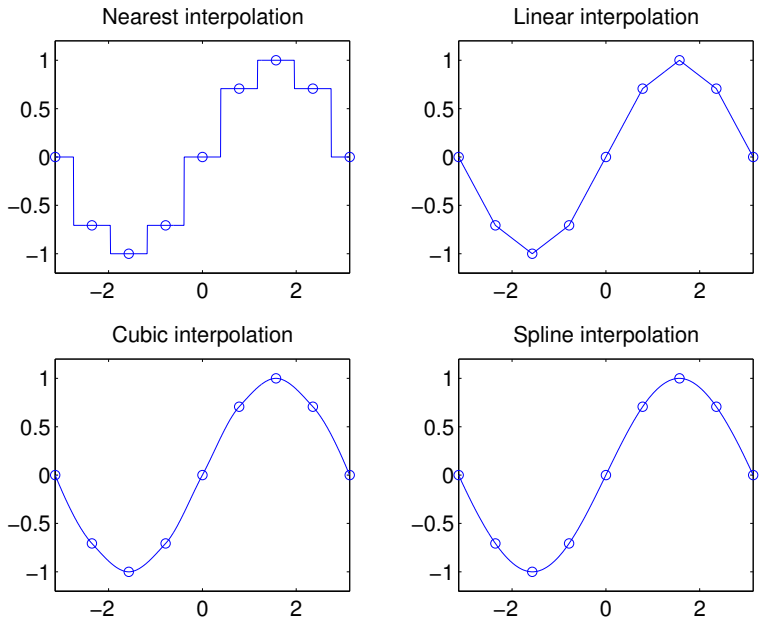
Dies wird hier am Beispiel der Sinus-Funktion erläutert:

```
xd = linspace(-pi,pi,9);
yd = sin(xd);

x  = linspace(-pi,pi,1000);
y  = sin(x);

y1 = interp1(xd,yd,x,'nearest');
y2 = interp1(xd,yd,x,'linear');
y3 = interp1(xd,yd,x,'cubic');
y4 = interp1(xd,yd,x,'spline');
```

Die Ergebnisse für die einzelnen Methoden sehen folgendermaßen aus:



Folgende Methoden stehen zur Verfügung:

nearest Nächste Nachbar Interpolation

linear Lineare Interpolation

cubic Interpolation mit Polynomen dritten Grades

spline Die Spline-Technik verwendet Polynome dritten Grades, wobei sichergestellt wird, dass sich sowohl die Werte als auch die ersten Ableitungen bei den Datenpunkten ein glattes Verhalten zeigen.

Solche Interpolationen stehen natürlich auch in höheren Dimensionen zur Verfügung. Siehe dazu die Hilfe zu den Funktionen `interp2` und `interp3`.