Institut für Physikalische Chemie

Methodenkurs „Anwendungen von Mathematica und Matlab in der Physikalischen Chemie“ im Sommersemester 2016

Prof. Dr. Stefan Weber, Dr. Till Biskup

— Lösungen zum Aufgabenblatt 7 zum Teil „Matlab“ vom 26.07.2016 —

Vorbemerkung

Die nachfolgend vorgestellten Lösungen wurden (unter MATLAB 2014a) getestet und sollten funktionieren. In den meisten Fällen gibt es aber mehr als eine Lösung, und die hier vorgestellte ist mit großer Wahrscheinlichkeit nicht die eleganteste oder beste und auf keinen Fall die „allein glückselig machende“.

Grundsätzlich gilt: Versuchen Sie immer, den Quellcode anderer Menschen in jedem Detail zu verstehen, bevor Sie ihn einsetzen. Nur so lernen Sie für sich selbst etwas dazu.

Aufgabe 7—1 (Polynomfit)

Die Anpassung von Polynomen *n*-ten Grades an Datenpunkte gehört zu den einfachsten in MATLAB durchführbaren Kurvenanpassungen. Das liegt, wie Sie noch bei Aufgabe 3 sehen werden, schlicht daran, dass sich diese Fragestellungen auf ein lineares Gleichungssystem zurückführen lassen, dessen Lösung bis auf eventuelle numerische Instabilitäten, die bestmöglich zu umschiffen MATLAB optimiert wurde, eindeutig ist und für die es bekannte Algorithmen gibt.

Zunächst müssen die in der Tabelle auf dem Aufgabenblatt gegebenen Datenpunkte in Vektoren in MATLAB umgewandelt werden. Die einfachste Möglichkeit ist die Erzeugung je eines ***x***- und ***y***-Vektors. Wenn Sie noch berücksichtigen, dass der ***x***-Vektor aus ganzen Zahlen besteht, sparen Sie sich etwas Tipparbeit:

Listing 1: Definition der beiden Vektoren *x* und *y* in MATLAB

```
x = (0:7);
y = [0.298 0.770 0.384 1.508 2.588 6.013 12.062 21.648];
```

Die Darstellung der Daten als unverbundene Punkte stellt nach der Bearbeitung des vorangegangenen Aufgabenblattes ebenfalls keinerlei Problem mehr dar.

Listing 2: Darstellung der beiden Vektoren *x* und *y* als unverbundene Symbole

```
plot(x,y,'ko');
```

Nun zum eigentlich interessanten Aspekt, der Anpassung von Polynomen ersten bis siebten Grades. MATLAB stellt Ihnen hier zwei bequem nutzbare Funktionen zur Verfügung, `polyfit` und `polyval`. Ersterer übergeben Sie den ***x***- und ***y***-Vektor und den Grad des gewünschten Polynoms und erhalten die Koeffizienten des Polynoms, letzterer die von `polyfit` zurückgegebenen Koeffizienten und den ***x***-Vektor

und erhalten das entsprechende Polynom evaluiert für die Elemente des Vektors $\boldsymbol{x}$. Für den einfachsten Fall der Anpassung eines Polynoms ersten Grades könnte das dann wie folgt aussehen:

Listing 3: Anpassung eines Polynoms ersten Grades in MATLAB

```
coeff = polyfit(x,y,1);
yfit  = polyval(coeff,x);
```

Da Sie Anpassungen für die Polynome ersten bis siebten Grades durchführen sollten, bietet es sich an, diese Anpassungen in einer Schleife durchzuführen und so Tipparbeit zu sparen. Einen vollständigen Quellcode für diese Aufgabe, der versucht, so allgemein wie möglich zu sein, finden Sie in nachfolgendem Listing.

Listing 4: Anpassung von Polynomen ersten bis siebten Grades an die gegebenen Datenpunkte

```
polydegree   = (1:7);
polynomials  = cell(length(polydegree),1);
coefficients = cell(length(polydegree),1);
legendText   = cell(length(polydegree)+1,1);
x2 = linspace(min(x),max(x));
colours = [0 0 1; 0 0.7 0; 1 0 0; .8 .8 0; 0 .7 .7; .7 0 .7; 0 0 0];
hold on;
for degree = 1:length(polydegree)
    coefficients{degree} = polyfit(x,y,polydegree(degree));
    polynomials{degree}  = polyval(coefficients{degree},x2);
    legendText{degree}   = sprintf('degree %i',degree);
    plot(x2,polynomials{degree},'Color',colours(degree,:));
end
legendText{end} = 'data points';
plot(x,y,'ko','MarkerFaceColor','k');
hold off;

xlabel('{\it x}');
ylabel('{\it y}');
legend(legendText,'Location','nw');
```

Zunächst definieren Sie sich einen Vektor mit den Graden der zu berechnenden Polynome (`polydegree`), anschließend werden die drei `cell arrays` für die Polynome, die Koeffizienten und den Legendentext vordefiniert. Da Sie zusätzlich zu den angepassten Polynomen auch noch die Datenpunkte einzeichnen wollen, ist das `cell array` für die Legende ein Element größer. Für die eigentliche Darstellung der angepassten Polynome brauchen Sie eine deutlich geringere Schrittweite als die der ursprünglichen Datenpunkte, weshalb es sich empfiehlt, hier einen eigenen $\boldsymbol{x}$-Vektor (`x2`) zu definieren. Zuletzt werden noch vor der Schleife Farben für die sieben darzustellenden angepassten Polynome definiert.

Die `for`-Schleife iteriert über die zuoberst definierten Grade der zu berechnenden Polynome und legt die Koeffizienten, die evaluierten Polynome und den Text der Legende jeweils in einem `cell array` zur späteren Verwendung ab. Anschließend wird das jeweils aktuelle Polynom über den `plot`-Befehl in der jeweiligen Farbe dargestellt. Nach der Schleife fügen Sie dem `cell array` für die Beschriftungen in der Legende noch ein Element für die darzustellenden Datenpunkte hinzu und stellen die Datenpunkte unverbunden ebenfalls dar. Es empfiehlt sich aufgrund der Vielzahl der Kurven in dieser Abbildung, Symbole zu wählen, die ausgefüllt dargestellt werden können, und diese Symbole *nach* all den Polynomen darzustellen, da sie so über den Kurven zu liegen kommen und besser sichtbar sind.

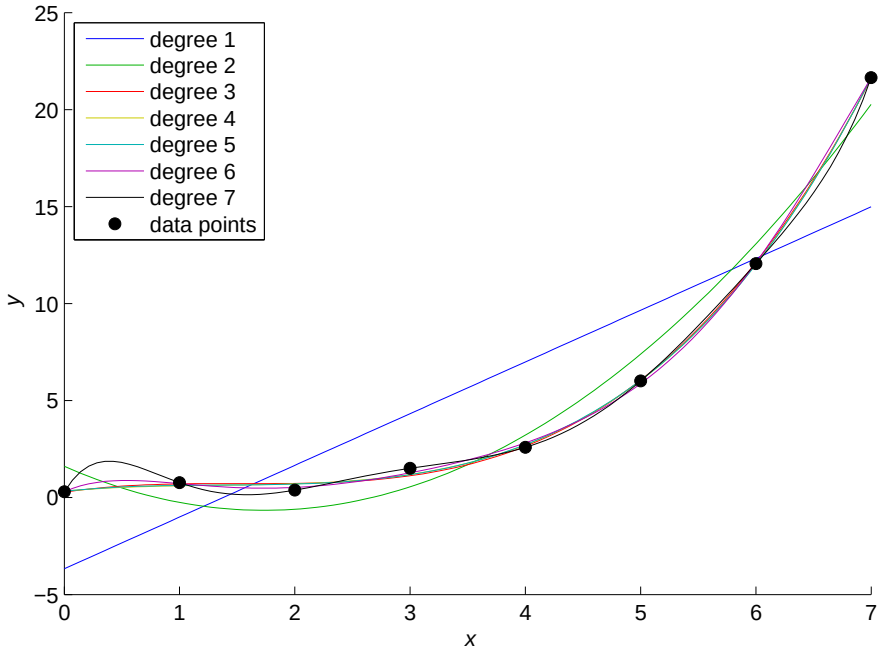

Abbildung 1: Grafische Auftragung der vorgegebenen Datenpunkte gemeinsam mit angepassten Polynomen ersten bis siebten Grades. Sie können auf den ersten Blick erkennen, dass die Polynome ersten und zweiten Grades die Datenpunkte nicht wirklich wiedergeben. Genauere Betrachtung der Polynome höheren Grades zeigt, dass ab dem sechsten Grad die Polynome „zu gut“ an die Datenpunkte angepasst werden. Den Quellcode für diese Abbildung finden Sie in Listing 4.

Das Ergebnis finden Sie in Abb. 1 dargestellt. Schon auf den ersten Blick wird ersichtlich, dass die Polynome ersten und zweiten Grades nicht geeignet sind, den Zusammenhang der Daten sinnvoll zu beschreiben. Ab dem Polynom sechsten Grades ist die Anpassung hingegen tendenziell „zu gut“, was sich darin äußert, dass die einzelnen Datenpunkte quasi exakt auf der Kurve des Polynoms liegen. Darüber hinaus kommen Sie hier in den Bereich, in dem sich der Grad des Polynoms der Anzahl der anzupassenden Datenpunkte annähert, Ihre Anpassung also tendenziell unterbestimmt ist.

Noch ein letztes Wort zur Speicherung der Koeffizienten in der Schleife in einem `cell array`: Während Sie für die Zwecke der Darstellung auch auf die Speicherung der Koeffizienten und des evaluierten Polynoms hätten verzichten können, ist zumindest die Ablage der Koeffizienten zum späteren Gebrauch sinnvoll, zumal Sie in einer realen Situation nicht nur eine möglichst gut angepasste Kurve darstellen wollen, sondern Interesse an den Koeffizienten des Polynoms haben.

Aufgabe 7—2 (Lineare Regression mit festem y-Achsenabschnitt)

Je nach zugrundeliegendem (physikalischem) Modell gibt es Situationen, in denen Sie nicht mit Hilfe der Funktionen `polyfit` und `polyval` ein Polynom anpassen können. Hier soll es aber gar nicht um Funktionen gehen, die komplizierter als ein Polynom sind, sondern um ein spezifisches Polynom ersten Grades: Eine Gerade mit festem y-Achsenabschnitt.

Das Problem bei der Verwendung von `polyfit` und `polyval` in diesem Kontext ist, dass Sie immer auch eine Anpassung des Koeffizienten nullten Grades, also des y-Achsenabschnittes bekommen. Abhilfe schafft hier, das lineare Gleichungssystem manuell aufzustellen und mit den von MATLAB zur Verfügung gestellten Möglichkeiten zu lösen. Wie das grundsätzlich geht, wurde Ihnen in der Präsentation zu dieser Lektion bereits vorgestellt.

Definieren Sie sich zunächst wieder aus den in der Tabelle gegebenen fehlerbehafteten Datenpunkten je

einen ***x***- und einen ***y***-Vektor und schauen Sie sich ggf. das Ergebnis als Darstellung untereinander nicht verbundener Datenpunkte an. Achten Sie gleich zu Anfang darauf, die x-Achse nicht erst bei der Eins, sondern bereits bei der Null beginnen zu lassen. Nur so können Sie später auf einen Blick sehen, ob Ihre lineare Regression korrekt ist und tatsächlich die Randbedingung $x(0) = 1$ erfüllt. Lassen Sie die y-Achse sinnvollerweise ebenfalls bereits bei Null beginnen.

Listing 5: Definition der beiden Vektoren x und y in MATLAB

```
x = (1:5);
y = [1.7937 2.5565 3.5307 4.4131 5.0879];
```

Mit der Aufstellung der beiden Vektoren ***x*** und ***y*** haben Sie bereits Ihr lineares Gleichungssystem aufgestellt, und zur Lösung dieses Gleichungssystems stehen Ihnen in MATLAB (mindestens) zwei Möglichkeiten zur Verfügung: der „Backslash"-Operator `\` bzw. der Befehl `lscov`. Welche Variante Sie verwenden wollen, hängt zum Einen von Ihrem persönlichen Geschmack ab, zum Anderen von Ihren Bedürfnissen. `lscov` kann Ihnen als zweiten Parameter noch Informationen zurückgeben, die Sie verwenden können, um die Güte der Anpassung zu quantifizieren. Details dazu finden Sie in der MATLAB-Dokumentation, z.B. via `doc lscov`.[1]

Da im vorliegenden Fall die Randbedingung nicht einfach $x(0) = 0$ lautet, sondern eben $x(0) = 1$, empfiehlt es sich zur Verallgemeinerung des Quellcodes, eine Variable `x0` mit dieser Randbedingung zu definieren, zumal Sie für das Lösen des Gleichungssystems von Ihrem ***y***-Vektor diesen Wert abziehen und ihn später beim Zeichnen der Ausgleichsgeraden wieder addieren müssen. Nachfolgend finden Sie den Quellcode für die beiden Varianten, dieses Gleichungssystem zu lösen:

Listing 6: Zwei Varianten zur Lösung des linearen Gleichungssystems in MATLAB

```
% Boundary condition
x0 = 1;

% Solving system of linear equations
a = x(:) \ (y(:)-x0);
a = lscov(x(:),y(:)-x0);
```

Beide Varianten arbeiten nur mit Spaltenvektoren. Die Vektoren ***x*** und ***y*** wurden oben (Listing 5) aber als Zeilenvektoren definiert. Eine bequeme Variante sicherzustellen, dass in jedem Fall Spaltenvektoren vorliegen, ist die Notation `x(:)` (bzw. `y(:)`), die hier zur Anwendung kam. So müssen Sie nicht überlegen, ob Sie Spalten- oder Zeilenvektoren definiert haben, und sind auf der sicheren Seite.

Zur Darstellung der Ausgleichsgeraden müssen Sie jetzt nur noch die (denkbar einfache) Geradengleichung $f(x) = ax + x(0)$ aufstellen und für mindestens zwei Punkte evaluieren. Auch hier kommt Ihnen wieder die oben allgemein definierte Variable `x0` für die Anfangsbedingung $x(0)$ zugute.

Listing 7: Evaluierung der Geradengleichung für zwei Punkte und Darstellung als Linie

```
x2 = [0 5.5];
plot(x2,a*x2+x0,'r-');
```

[1] Die Bestimmung der Güte einer Anpassung eines funktionalen Zusammenhangs an Daten ist ein zentraler Aspekt der Kurvenanpassung bzw. Regression, auf den im Rahmen dieses Kurses nicht eingegangen wird. In der praktischen Anwendung sollten Sie aber nie darauf verzichten. Weite Informationen finden Sie in der einschlägigen Literatur, für den Geradenausgleich u.a. in einführenden Abhandlungen zu den diversen Grundpraktika in den Naturwissenschaften.

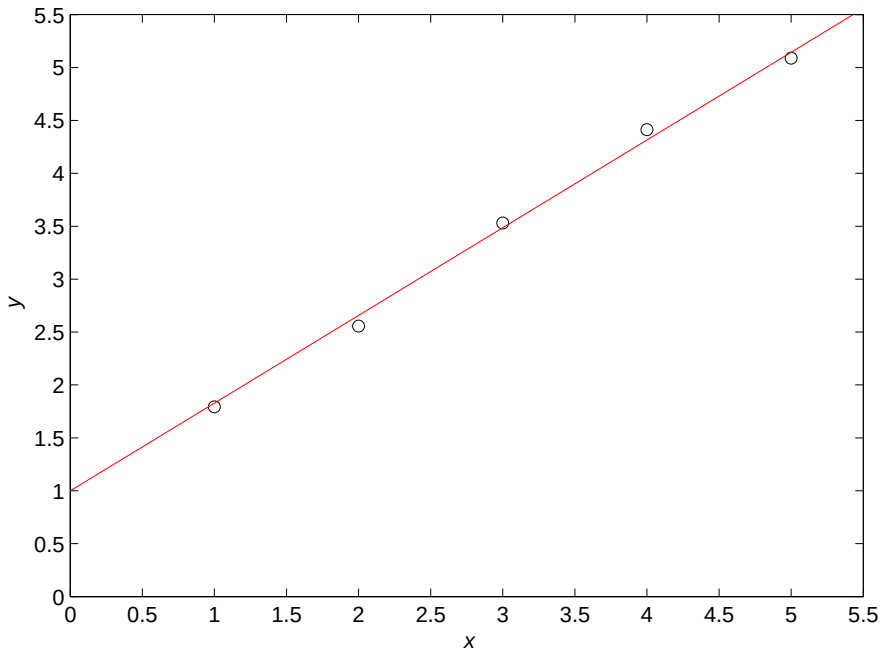

Abbildung 2: Grafische Auftragung der vorgegebenen Datenpunkte gemeinsam mit einer linearen Regression mit Randbedingungen. Angepasst wurde eine Gerade der Form $f(x) = ax+x(0)$ mit der Randbedingung $x(0) = 1$. Den Quellcode für diese Abbildung finden Sie in Listing 8.

Als $\boldsymbol{x}$-Vektor für die Gerade wurden hier zwei Punkte gewählt, da Sie bekanntlich durch zwei Punkte eine Gerade vollständig definieren können und MATLAB ohne explizite Angabe Punkte im `plot`-Befehl miteinander verbindet. Die Grenzen für den $\boldsymbol{x}$-Vektor können Sie ggf. auch direkt von der aktuellen x-Achse abfragen, wenn Sie die Datenpunkte bereits in einer Abbildung dargestellt haben. Der Befehl dafür lautete: `get(gca,'XLim')`. Sie erhalten einen Zwei-Element-Vektor als Rückgabewert.

Die grafische Auftragung der linearen Regression der gegebenen Datenpunkte mit der Anfangsbedingung $x(0) = 1$ finden Sie in Abb. 2, den vollständigen Quellcode, der zu dieser Abbildung führte, in nachfolgendem Listing:

Listing 8: Vollständiges Beispiel einer linearen Regression mit Anfangsbedingungen

```
x = (1:5);
y = [1.7937 2.5565 3.5307 4.4131 5.0879];

plot(x,y,'ko');
xlim([0 5.5]);
ylim([0 5.5]);

x0 = 1;
a = lscov(x(:),y(:)-x0);
x2 = get(gca,'XLim');
hold on;
plot(x2,a*x2+x0,'r-');
hold off;

xlabel('{\it x}');
ylabel('{\it y}');
```

Aufgabe 7—3 (Lineare Regression komplexerer Funktionen)

Das Konzept der linearen Regression ist nicht auf Geraden und Polynome beschränkt, sondern lässt sich auf alle Funktionen anwenden, die nur linear von ihren Koeffizienten abhängen (daher auch der Name). Der große Vorteil der linearen Regression gegenüber nichtlinearen Kurvenanpassungen ist, dass lineare Gleichungssysteme in der Regel eine eindeutige Lösung aufweisen und die Algorithmen zu ihrer Lösung bekannt und numerisch relativ robust sind.

Ein Beispiel für eine Funktion, die sich so umschreiben lässt, dass sie nur noch linear von ihren Koeffizienten abhängt, ist die Ihnen in dieser Aufgabe gegebene Funktion

$$y = A \cdot \sin(x + \phi)$$

die Sie als Funktion zweier Parameter umschreiben können. Dazu müssen Sie sich lediglich der Tatsache bewußt sein, dass sich Sinus und Kosinus nur in ihrer Phase unterscheiden und also entsprechend ineinander überführt werden können. So kommen Sie zu der zur oben gegebenen Gleichung äquivalenten Formulierung

$$y = a_1 \sin(x) + a_2 \cos(x)$$

die sich als lineares Gleichungssystem formulieren und wie gehabt in MATLAB lösen lässt, um die Koeffizienten a_i zu erhalten.

Um die Ihnen gegebene Funktion $f(x) = \frac{\pi}{2} \sin(x + \frac{2\pi}{3})$ in MATLAB zu implementieren und anschließend darzustellen, können Sie wieder auf die Möglichkeiten anonymer Funktionen zurückgreifen oder aber die Funktion direkt in Abhängigkeit von x definieren. Eine Umsetzung könnte wie folgt aussehen:

Listing 9: Darstellung der trigonometrischen Funktion in MATLAB

```
x = linspace(0,20,100);
y = pi/2*sin(x+2*pi/3);
```

Der hier verwendete Befehl `linspace` erzeugt Ihnen einen Vektor äquidistanter Werte in den als erste beide Parameter übergebenen Grenzen. Der dritte (optionale) Parameter legt die Zahl der Elemente fest, die der resultierende Vektor enthalten soll.

Als Nächstes müssen Sie das lineare Gleichungssystem aufstellen und lösen. Dazu definieren Sie sich zunächst die Vektoren für die einzelnen Terme, die jeweils nur linear von ihren Koeffizienten abhängen, und fassen diese Vektoren in einer gemeinsamen Matrix zusammen. Wichtig ist hier, dass die Vektoren für die einzelnen Terme die Spalten der resultierenden Matrix $\boldsymbol{F}$ bilden. Auch hier können Sie wieder von der speziellen Notation in MATLAB Gebrauch machen, die Ihnen sicherstellt, mit Spaltenvektoren zu operieren. Das Ergebnis der Anspassung erhalten Sie einfach durch Multiplikation Ihrer Matrix $\boldsymbol{F}$ mit dem die Koeffizienten enthaltenden Vektor $\boldsymbol{a}$.

Listing 10: Aufstellung des linearen Gleichungssystems und seine Lösung

```
F1 = sin(x);
F2 = cos(x);
F  = [F1(:),F2(:)];
a  = F\y(:);

yFit = F*a;
```

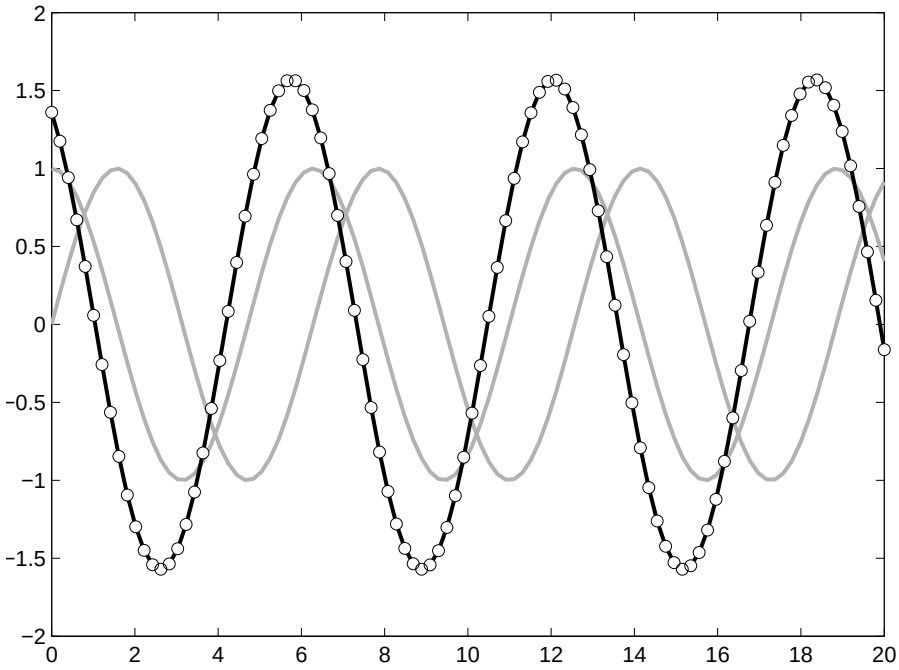

Abbildung 3: Lineare Regression komplexerer Funktionen. Die offenen Kreise repräsentieren die originale Funktion, die grauen Linien die beiden zugrundeliegenden Funktionen, Sinus und Cosinus, und die schwarze Linie das Resultat der Kurvenanpassung mittels linearer Regression. Den MATLAB-Code zur Erzeugung dieser Abbildung finden Sie in Listing 12.

Sind Sie nun noch an den beiden Parametern interessiert, die die ursprüngliche Gleichung charakterisierten, nämlich der Amplitude A und der Phase ϕ, dann müssen Sie ein wenig Vorwissen bzgl. der trigonometrischen Funktionen mitbringen. Die entsprechenden Parameter berechnen sich in MATLAB aus den Koeffizienten a_i wie folgt:

Listing 11: Berechnung der Parameter der ursprünglichen Funktion in MATLAB

```
amplitude = norm(a);
phase = atan2(a(2),a(1));
```

Um das Ergebnis Ihrer linearen Regression entsprechend gemäß der Vorgaben darzustellen, können Sie sich Ihres bei der Bearbeitung des vorausgegangenen Aufgabenblattes erworbenen Wissens zu grafischen Darstellungen in MATLAB bedienen. Ein Beispiel ist Ihnen nachfolgend angegeben:

Listing 12: Darstellung der Ausgangsdaten, der beiden linearen Terme und der angepassten Kurve

```
gray = 0.7*ones(3,1);
plot(x,F1,x,F2,'Color',gray,'LineWidth',2);
hold on;
plot(x,yFit,'k','LineWidth',2);
plot(x,y,'ko','Markerfacecolor','w');
hold off;
```

Um zu den Ausgangsdatenpunkten Rauschen hinzuzuaddieren, können Sie von der Funktion `rand` Gebrauch machen, die Ihnen normalverteiltes Rauschen im Intervall $[0, 1]$ erzeugt. Achten Sie darauf, dieses Rauschen symmetrisch um die Null zu machen.

Eine mögliche Erzeugung eines alternativen, verrauschten $\boldsymbol{y}$-Vektors, die in ihrer Implementierung unabhängig von der tatsächlichen Größe von $\boldsymbol{x}$ ist, wäre:

Listing 13: Erzeugung eines Datenvektors mit aufaddiertem, symmetrischem Rauschen

```
y = pi/2*sin(x+2*pi/3)+(rand(size(x))-0.5)*.5;
```

Sicherlich können Sie hier mit der Amplitude des Rauschens ein wenig spielen. Interessant wäre in diesem Zusammenhang, sich die durch Lösung des nichtlinearen Gleichungssystems angepassten Koeffizienten neben den ursprünglich zur Erzeugung der Daten verwendeten ausgeben zu lassen. Das erlaubte Ihnen eine Abschätzung der Güte der Anpassung je nach Rauschniveau. Diese Strategie ist sehr nützlich, wenn Sie die Robustheit einer Anpassung gegenüber verrauschten Daten testen wollen. Schließlich haben Sie es in der Realität nicht immer mit quasi rauschfreien Daten zu tun.

Aufgabe 7—4 (Nichtlineare Kurvenanpassung beliebiger Funktionen)

Nur eine eher kleine Gruppe von Funktionen lässt sich so umschreiben, dass sie nur linear von ihren Koeffizienten abhängen. Diese Funktionen lassen sich entsprechend auch nicht als lineare Gleichungssysteme formulieren, und in der Regel gibt es auch keine eindeutige Lösung, bzw. es gibt kein analytisches Verfahren, das das globale Minimum der Anpassung (also die bestmögliche Anpassung des Modells an die Daten) mit Sicherheit bestimmen kann.

Das generelle Vorgehen bei der Anpassung nichtlinearer Funktionen ist oft gleich: Ziel ist die Minimierung der Abweichung zwischen nichtlinearer Funktion und Daten, und ein gängiges Maß für die Güte der Anpassung ist die Summe der Quadrate der Abweichungen. Die Abweichungen werden quadriert, da sie grundsätzlich vorzeichenbehaftet sind.

Welche Verfahren zur Minimierung konkret zum Einsatz kommen, hängt von der Fragestellung und den verfügbaren Implementierungen ab. In MATLAB sind die bekannteren Algorithmen wie Gauß-Newton oder Levenberg-Marquardt nicht im Standard-Umfang enthalten, sondern nur in speziellen Toolboxen. Der Algorithmus, der in der hier verwendeten Funktion `fminsearch` implementiert ist, die zum Standard-Funktionsumfang von MATLAB gehört, ist der Nelder-Mead-Simplex-Algorithmus. Die Erfahrung zeigt, dass er mitunter relativ empfindlich gegenüber den Anfangsbedingungen ist. Trotzdem können Sie ihn vielfältig einsetzen, ohne gleich teure zusätzliche Toolboxen in MATLAB nutzen oder eigene Minimierungsalgorithmen implementieren zu müssen.

Das Laden und Darstellen der zur Aufgabe gehörenden Daten gestaltet sich einfach, zumal die Daten bewusst als reine ASCII-Daten abgelegt wurden, die Sie bequem mit dem Befehl `load` in MATLAB importieren können. Der Fokus liegt hier ja schließlich auf der nichtlinearen Kurvenanpassung, nicht auf dem Datenimport.

Listing 14: Einlesen und erste Darstellung der fehlerbehafteten Daten

```
data = load('noisyData.txt');
x=data(:,1);
y=data(:,2);
plot(x,y,'ko');
```

Wie in der Aufgabenstellung schon angesprochen, müssen Sie der Funktion `fminsearch` schon die zu minimierende Funktion übergeben, also an unserer Stelle die Summe der Quadrate der Abweichungen zwischen anzupassender Funktion und Daten. Deshalb ist es hilfreich, wenn Sie sich zwei anonyme Funktionen definieren, zum einen die anzupassende Funktion (hier die Lorentz-Funktion), zum anderen die zu mininierende Funktion, also die Summe der Quadrate der Abweichungen. Beide Funktionen sollten Sie in Abhängigkeit vom Vektor $\boldsymbol{a}$ der Koeffizienten definieren. Das könnte dann wie folgt aussehen:

Listing 15: Anonyme Funktionen

```
plotfun = @(a)a(1)./((x-a(2)).^2+a(3));
fitfun = @(a)sum((y-plotfun(a)).^2);
```

Ein großer Vorteil der Verwendung anonymer Funktionen gegenüber dem Schreiben von Funktionen in eigenen Dateien ist in diesem Fall, dass anonyme Funktionen auf alle lokal verfügbaren Variablen, im vorliegenden Fall also `x` und `y`, zugreifen können, was Vieles vereinfacht. Haben Sie komplexere Modelle, die Sie an Ihre Daten anpassen möchten, können Sie allerdings nicht mehr auf dieses Mittel zurückgreifen, sondern müssen andere Wege finden, einer zu minimierenden Funktion weitere Parameter über den Vektor mit Koeffizienten hinaus mitzugeben. Auch dafür gibt es Strategien, die aber weit über das hinausgehen, was im Rahmen dieses Kurses sinnvoll behandelt werden kann. Sie finden aber entsprechende Hilfestellung in der MATLAB-Dokumentation.

Nachdem Sie die zu minimierende Funktion `fitfun` definiert haben, müssen Sie nur noch den Koeffizienten-Vektor $\boldsymbol{a}$ mit Startwerten definieren und beide dann gemeinsam der Funktion `fminsearch` zur Minimierung übergeben.

Listing 16: Aufruf von `fminsearch` mit dem Koeffizientenvektor und der zu minimierenden Funktion

```
a = [70 50 100];
fita = fminsearch(fitfun,a);
```

Für die Darstellung können Sie jetzt großen Nutzen aus der getrennten Definition der darzustellenden (bzw. eigentlich der anzupassenden) Funktion `plotfun` als Funktion des Koeffizienten-Vektors ziehen. Sowohl mit den Startwerten als auch mit den angepassten Koeffizienten ist das jeweils nur ein einzelner Funktionsaufruf:

Listing 17: Grafische Darstellung der anzupassenden Funktion mit Startparametern und angepassten Koeffizienten

```
hold on;
plot(x,plotfun(fita),'k-','LineWidth',2);
plot(x,plotfun(a),'k--','LineWidth',2);
hold off;
```

Die zusätzlichen Angaben zu Linienstil und Liniendicke dienen dazu, die Kurven vor dem Hintergrund der doch relativ stark verrauschten Datenpunkte etwas deutlicher hervorzuheben. Das Ergebnis finden Sie in Abb. 4 dargestellt.

Wenn Sie die angepassten Koeffizienten mit den zur Erzeugung der Datenpunkte verwendeten Koeffizienten $\boldsymbol{a}_{\text{orig}} = (20, 30, 30)$ vergleichen, werden Sie feststellen, dass Sie trotz eines ganz erheblichen Rauschens – der Signal-Rausch-Abstand, den Sie sich auch relativ bequem selbst mit MATLAB berechnen können, ist deutlich kleiner als zwei – eine recht gute Kurvenanpassung mit einem maximalen Fehler der angepassten Koeffizienten von $< 12\%$ erzielt haben.

Zur Bestimmung des Signal-Rausch-Abstandes: Eine Strategie wäre, Minimum und Maximum des Bereiches der Datenpunkte zwischen ca. 60 und 100 zu bestimmen, die Hälfte davon vom globalen Maximum des Datenvektors abzuziehen und dieses so „rauschkorrigierte“ Maximum ins Verhältnis zur Rauschamplitude zu setzen. Eine verlässlichere Variante, die Rauschamplitude zu bestimmen, die weniger anfällig für „Ausbrecher“ in den Datenpunkten ist, stützt sich auf die Standardabweichung des Bereiches Ihrer Daten, den Sie als Rauschen deklariert haben. Sie finden den Signal-Rausch-Abstand gerade in der Spektroskopie

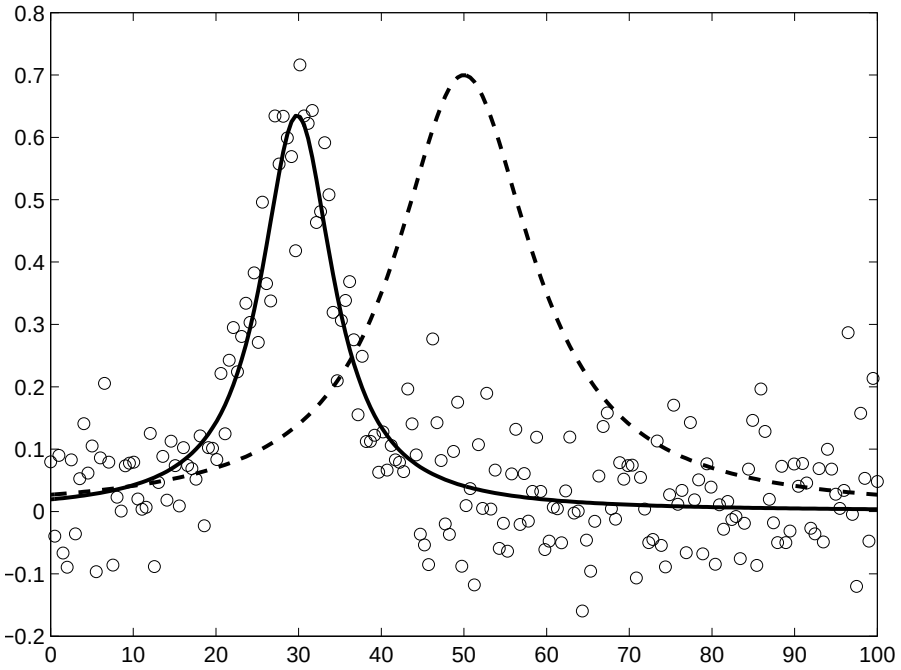

Abbildung 4: Nichtlineare Regression verrauschter Datenpunkte. Die offenen Kreise repräsentieren die verrauschten Daten, die gestrichelte schwarze Linie die Lorentz-Funktion mit den Startwerten, die durchgezogene schwarze Linie das Resultat der Kurvenanpassung mittels nichtlinearer Regression. Den zur Darstellung verwendeten MATLAB-Quellcode finden Sie in Listings 14–17.

häufig definiert als Verhältnis von Signalamplitude zu Standardabweichung des Rauschens. Beachten Sie auch hier wieder, dass Ihre Signalamplitude ebenfalls rauschbehaftet ist und Sie jeweils betragsmäßig die halbe Rauschamplitude von der Signalamplitude abziehen sollten.

Noch ein Wort zur Praxis: Da der Algorithmus, der in `fminsearch` zum Einsatz kommt, nicht übermäßig robust ist, müssen Sie ggf., wenn Ihnen gute Startwerte für Ihre anzupassenden Koeffizienten nicht bekannt sind, diese Startwerte über einen (physikalisch oder anderweitig) sinnvollen Bereich streuen und wiederholte Versuche der Kurvenanpassung von diesen unterschiedlichen Startpunkten in der Parameterlandschaft aus starten. Diese Herangehensweise liefert Ihnen gleichzeitig einen Zugang zur Robustheit Ihrer Anpassung.

Es sei hier nicht verschwiegen, dass die Anpassung nichtlinearer Modelle an (gemessene) Daten meist ein nichttriviales Problem und gleichzeitig Gegenstand aktiver Grundlagenforschung ist. Ein hier gar nicht behandelter, wenngleich sehr wichtiger, Aspekt ist darüber hinaus die Quantifizierung der Güte Ihrer Anpassungen. Die meisten in MATLAB implementierten Funktionen liefern Ihnen entsprechend (optional) zusätzliche Informationen, die Sie für die Berechnung der Güte verwenden können. Für Details seien Sie auf die einschlägige Literatur verwiesen.