Institut für Physikalische Chemie

Methodenkurs „Anwendungen von Mathematica und Matlab in der Physikalischen Chemie“ im Wintersemester 2017/2018

Prof. Dr. Stefan Weber, Dr. Till Biskup

— Lösungen zum Aufgabenblatt 6 zum Teil „Matlab“ vom 27.02.2018 —

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 6—1 (Mehrere Plots in einem Fenster)

Der `plot`-Befehl in MATLAB ist beides, sehr bequem, aber auch „unter der Haube“ sehr aufwendig. Der Nutzer muss sich z.B. nicht darum kümmern, ob es schon ein Grafikfenster gibt, und auch die Achsen werden weitestgehend automatisch an die darzustellenden Daten angepasst. Dafür hat der `plot`-Befehl den „Schönheitsfehler“, die bisherigen Inhalte des aktuellen Grafikfensters kommentarlos zu löschen, so es existiert.

Eine häufig anzutreffende Anforderung ist aber, mehr als eine Linie gleichzeitig in einer Grafik darzustellen. Um dem gerecht zu werden, gibt es (mindestens) zwei vergleichsweise einfache Strategien: entweder Sie übergeben einem `plot`-Befehl mehr als ein Paar von Vektoren (ein Paar für jede zu zeichnende Linie), oder Sie verwenden das Befehlspaar `hold on` und `hold off`.

Zur Untersuchung und Verdeutlichung des unterschiedlichen Verhaltens von MATLAB, je nachdem, welche dieser Strategien Sie verfolgen, sollten Sie zunächst die Funktionen $y1(x)$, $y2(x)$ und $y3(x)$ in MATLAB definieren und anschließend in einem vorgegebenen Intervall mit einer sinnvollen Schrittweite darstellen. Auch hier gibt es wieder mehrere Möglichkeiten der Implementierung. Wollen Sie zunächst die Funktionen definieren und sich anschließend über das Intervall und die Schrittweite Gedanken machen, empfiehlt sich der Weg über anonyme Funktionen.

Listing 1: Erzeugung der darzustellenden Daten als Inline-Funktionen

```
y1 = @(x)sin(x)+sin(3*x);
y2 = @(x)sin(x).^2+1/2*sin(3*x);
y3 = @(x)sin(x).^3+1/3*sin(3*x);
```

Anderenfalls können Sie auch erst den Vektor $\boldsymbol{x}$ im angegebenen Intervall mit entsprechenden Zwischenschritten definieren und anschließend die drei Vektoren $\boldsymbol{y}_1$, $\boldsymbol{y}_2$ und $\boldsymbol{y}_3$ damit erzeugen. Was Sie als „sinnvolle“ Zwischenschritte für den Vektor $\boldsymbol{x}$ bezeichnen, ist letztlich eine Frage persönlicher Vorlieben. Ziel wäre hier die glatte Darstellung der Kurven ohne sichtbare Stufen.

Listing 2: Direkte Erzeugung der darzustellenden Daten

```
x1 = 0:0.1:12;

y1 = sin(x1) + sin(3*x1);
y2 = sin(x1).^2 + sin(3*x1)./2;
y3 = sin(x1).^(3) + sin(3*x1)./3;
```

Die beiden Möglichkeiten, diese drei Funktionen in einer Abbildung darzustellen, sind wie bereits angemerkt die Verwendung eines einzigen `plot`-Befehls mit drei Paaren an ***x***- und ***y***-Vektoren,

Listing 3: Darstellung mehrerer Linien in einer Abbildung durch einen `plot`-Befehl

```
plot(x,y1,x,y2,x,y3);
```

oder die Nutzung der Befehle `hold on` und `hold off`, wobei der erstere bereits vor dem ersten Aufruf von `plot` erfolgen kann, spätestens aber direkt vor dem zweiten Aufruf von `plot`, und der letzte nach dem letzten `plot`-Befehl stehen (und nach Möglichkeit nicht vergessen werden) sollte.

Listing 4: Darstellung mehrerer Linien in einer Abbildung mit je einem `plot`-Befehl

```
hold on;
plot(x,y1);
plot(x,y2);
plot(x,y3);
hold off;
```

Der Unterschied zwischen den beiden oben aufgeführten Varianten wird Ihnen sofort offensichtlich, wenn Sie sie selbst ausprobieren: Bei Nutzung eines einzigen `plot`-Befehls wechselt MATLAB automatisch die Linienfarben und Stile durch. Wenn Sie hingegen innerhalb von `hold on` und `hold off` mehrere `plot`-Befehle ausführen, ohne explizit eine Linienfarbe oder einen Linienstil anzugeben, erscheinen alle Linien blau und durchgezogen.

Es steht Ihnen natürlich frei, bei jedem `plot`-Befehl die Linienfarbe und den Linienstil explizit anzugeben. Darüber hinaus können Sie auch bei einer bereits bestehenden Abbildung durch die Befehle `hold on` und `hold off` weitere Linien hinzufügen, deren Erscheinungsbild Sie über optionale Parameter komplett kontrollieren können. Insofern sind Sie mit dieser zweiteren Variante ggf. flexibler.

Aufgabe 6—2 (Achsenbeschriftungen inklusive Formatierung)

Die beiden einfachsten Befehle zur Beschriftung der Achsen lauten `xlabel` und `ylabel`. Für drei Achsen gäbe es dann entsprechend noch eine Funktion `zlabel`. Im einfachsten Falle übergeben Sie diesen Funktionen jeweils eine Zeichenkette mit dem Text, der als Achsenbeschriftung in der Abbildung erscheinen soll. Dabei wird immer die gerade aktive Achse (`gca`) der gerade aktiven Abbildung (`gcf`) bearbeitet. Die Zeichenketten durch Beschriftung verstehen – wie viele Beschriftungen von Grafikobjekten in MATLAB – bis zu einem gewissen Grad LaTeX-Steuerzeichen, also z.B. `\it` für den kursiven Textsatz. Um die Wirkung dieser Steuercodes auf einen bestimmten Bereich zu beschränken (wie das für die korrekte Beschriftung von Achsen oft notwendig ist, wo die Größe kursiv, die Einheit aber aufrecht gesetzt wird), verwenden Sie geschweifte Klammern.

Um gemäß der Aufgabenstellung die bei der Bearbeitung der vorangegangenen Aufgabe erzeugten Abbildungen mit korrekten Achsenbeschriftungen zu versehen, sähen die beiden Befehle also wie folgt aus:

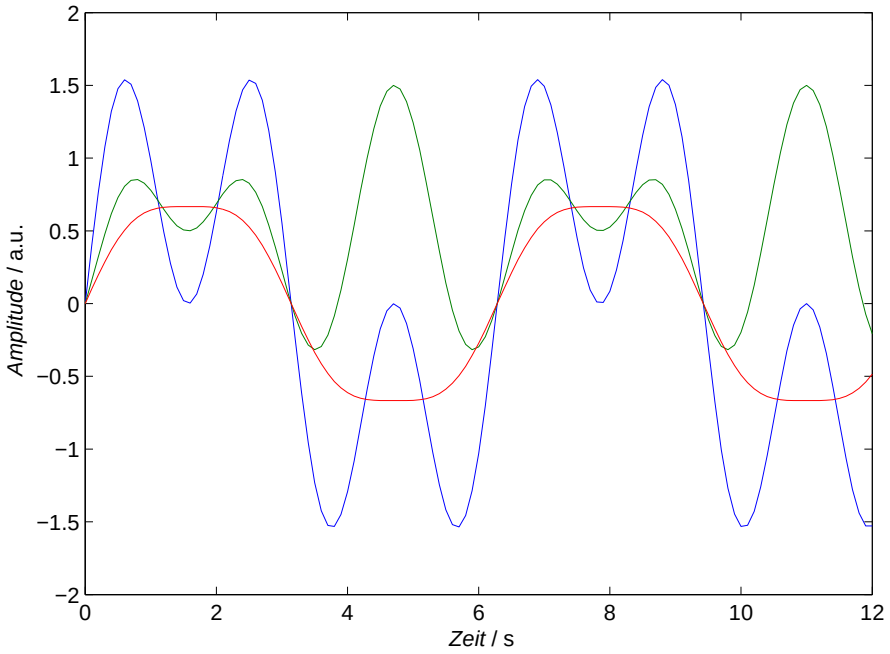

Abbildung 1: Grafische Auftragung der drei angegebenen Funktionen mit einem Plotbefehl. Wie Sie erkennen können, wechselt MATLAB automatisch die Linienfarben durch. Die Reihenfolge ist dabei rot-grün-blau-cyan-magenta-gelb-schwarz (b-g-r-c-m-y-k). Anschließend werden dann die Linienstile durchgewechselt, auch wenn mehr als sieben Linien in einer Darstellung nur in Ausnahmen hilfreich sind und nicht der Übersichtlichkeit schaden.

Listing 5: Einfachste Form der wissenschaftlich korrekten Achsenbeschriftungen

```
xlabel('{\it Zeit} / s')
ylabel('{\it Amplitude} / a.u.')
```

Wollten Sie eine andere als die gerade aktive Achse bearbeiten, können Sie entweder die aktive Achse wechseln (beispielsweise bei einem `subplot` durch Aufruf des jeweiligen Bereichs) oder aber, so Sie Zugriff auf die Referenz (das *handle*) der gewünschten Achse in einer Variable haben, diese Variable als erstes Argument den Befehlen zum Setzen der Achsenbeschriftung mitgeben. Darüber hinaus können Sie diesen Befehlen wie den meisten Grafik-Befehlen von MATLAB noch eine ganze Reihe an Schlüssel-Wert-Paaren als optionale Parameter übergeben und damit detailliert das Aussehen und Verhalten kontrollieren. Erinnern Sie sich, dass Grafiken in MATLAB letztlich Objekte sind. Objekte sind Strukturen (`struct`) nicht unähnlich, und so wie Strukturen Felder haben, so haben Objekte Eigenschaften, die Sie mit Namen ansprechen können. Details dazu finden Sie in der MATLAB-Hilfe zu den jeweiligen Grafik-Befehlen. Das Ergebnis der bisherigen `plot`-Befehle sehen Sie in Abb. 1.

Aufgabe 6—3 (Linienstile und Symbole)

MATLAB stellt Ihnen eine ganze Reihe an Linienstilen und Symbolen zur Verfügung. Während die Zahl der Linienstile eher überschaubar bleibt (durchgezogen, gestrichelt, gepunktet, gestrichpunktet), ist die Zahl der Symbole dann doch deutlich höher. Eine tabellarische Übersicht finden Sie auf den Folien zur Lektion oder in der MATLAB-Hilfe. Hinsichtlich der Farben sind Ihnen (fast) keine Grenzen gesetzt, da Sie Farben in MATLAB über RGB-Vektoren (für die drei Grundfarben rot, grün und blau) selbst definieren können. Auch die Liniendicke ist weitgehend vom Nutzer einstellbar. Es gibt eine Kurzschreibweise für den Linienstil, die Stil, Farbe (aus den sieben oben genannten Farben und Weiß) und Symbol als einen

String zusammenfasst. Die Liniendicke müssen Sie allerdings über eine Schlüssel-Wert-Zuweisung mit dem Schlüssel `LineWidth` einstellen.

Um die in der Aufgabenstellung verlangte Darstellung mit drei unterschiedlichen Linienstilen zu verwirklichen, gibt es wiederum viele unterschiedliche Möglichkeiten. Entweder Sie geben jeweils direkt beim `plot`-Befehl die entsprechenden Linienspezifikationen an,

Listing 6: Angabe des Linienstils direkt im `plot`-Befehl

```
plot(...
    x,y1,'b-',...
    x,y2,'g--',...
    x,y3,'r:' ...
    );
```

oder aber Sie ändern ganz oder teilweise nach der erfolgen Darstellung die Eigenschaften einzelner Linien über den Befehl `set`. Letzteres ist Ihnen (mit identischer Wirkung wie oben bei der direkten Angabe im `plot`-Befehl) für die hier gefragte Darstellung gezeigt.

Listing 7: Angabe des Linienstils direkt im `plot`-Befehl

```
h=plot(x,y1,x,y2,x,y3);
set(h,{'LineStyle'},{'-';'--';':'})
```

Beachten Sie beim letzten Beispiel, dass Sie sich eine Referenz auf die Linien des `plot`-Befehls zurückgeben lassen müssen, die Sie dann wiederum als erstes Argument für den `set`-Befehl verwenden können. Genauer gesagt handelt es sich hier um eine Liste von Referenzen, für jede Linie eine.

Je nach verwendeter MATLAB-Version werden Sie feststellen, dass das Grün, dass Sie mit der Angabe des Kürzels „`g`“ erhalten, nicht dem Grün entspricht, das MATLAB verwendet, wenn es automatisch durch die Linienfarben iteriert. Da es sich bei der Angabe des Kürzels „`g`“ um den RGB-Wert $(0, 1, 0)$ handelt, ist das meist ein zu helles Grün, sowohl für den Bildschirm (besonders vorsichtig sollten Sie bei Präsentationen mit derlei Farben sein) als auch im Druck.

Gerade was die Linienbreiten im Zusammenhang mit von der durchgezogenen Linie abweichenden Linienstilen angeht, gibt es mitunter massive Unterschiede im Aussehen zwischen dem Abbildungsfenster in MATLAB und dem Ergebnis einer exportierten Datei, insbesondere wenn Sie als Exportformat das (aufgrund seiner Vektorisierung) meist zu empfehlende PDF-Format wählen. Auf der anderen Seite ist es mitunter empfehlenswert, für den Export von Abbildungen (gerade im Hinblick auf Präsentationen) die Linienbreite von ihrem Standardwert (0.5 pt) etwas hochzusetzen (1–1.5 pt).

Abschließend sei Ihnen noch ein Beispiel für die Verwendung diverser Symbole gezeigt, inklusive der unterschiedlichen Möglichkeiten, diese Eigenschaften einer Linie zu kontrollieren.

Listing 8: Angabe des Linienstils direkt im `plot`-Befehl

```
x = 0:0.15:12;

%...

plot(x,y1,'ok-');
plot(x,y2,'sb');
plot(x,y3,'Marker','d','MarkerFaceColor','r','MarkerEdgeColor','k',...
    'MarkerSize',9,'LineWidth',2,'LineStyle','none');
```

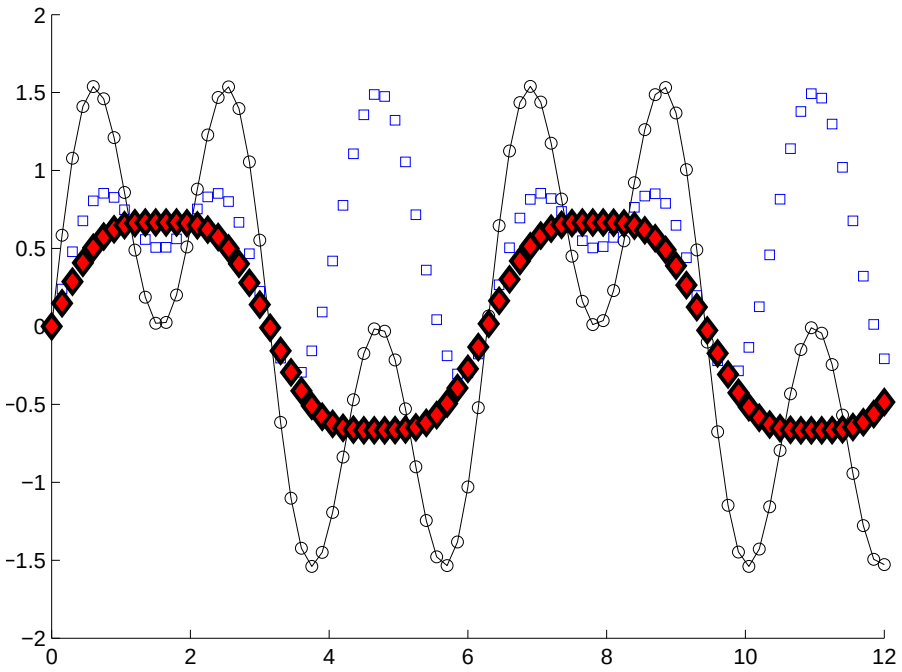

Abbildung 2: Grafische Auftragung der drei in der ersten Aufgabe angegebenen Funktionen mit unterschiedlicher Liniengestaltung. Den MATLAB-Quellcode zur Erzeugung dieser Abbildung finden Sie in Listing 8. Beachten Sie, dass für diese Darstellung die Schrittweite etwas vergrößert wurde, damit die Symbole weiter auseinander sind und Sie klar erkennen können, in welchen Fällen eine durchgezogene Linie die Symbole verbindet.

Das Ergebnis dieser drei Zeilen sehen Sie in Abb. 2 dargestellt. Es erhebt explizit keinen Anspruch auf Ästhetik, sondern soll Ihnen nur einige der Möglichkeiten aufzeigen, die Ihnen MATLAB bietet. Insbesondere sehen Sie in den ersten beiden Zeilen, eine elegante Kurzschreibweise, um (offene) Symbole in einer bestimmten Farbe mit oder ohne (gleichfarbige) Linie zu erzeugen. Der dritte `plot`-Befehl ist hingegen ein Beispiel dafür, wie Sie die einzelnen Eigenschaften einer Linie durch hintereinander gereihte Schlüssel-Wert-Zuweisungen detailliert kontrollieren können. Beachten Sie, dass sich die Linienbreite (`LineWidth`) auch auf die Umrandung der Symbole auswirkt. Die Schreibweise der Schlüssel hinsichtlich Groß- und Kleinschreibung ist beliebig, die hier verwendete Variante entspricht der Definition in der MATLAB-Hilfe und führt zu einer besseren Lesbarkeit.

Der vielleicht für die Anwendung in den Naturwissenschaften wichtigste Aspekt: Wollen Sie Datenpunkte nur als (offene) Symbole darstellen, ohne sie mit einer Linie zu verbinden, ist das im Zweifelsfall denkbar einfach: Sie müssen nur als drittes (optionales) Argument eines `plot`-Befehls das Kürzel für das jeweilige Symbol als Zeichenkette übergeben.

Aufgabe 6—4 (Legenden)

Ein weiteres wesentliches Merkmal (wissenschaftlicher) Abbildungen sind Legenden, zumal wenn mehr als eine Linie dargestellt ist und dem Betrachter ein schneller Überblick gegeben werden soll. Bedeutung kommt Legenden darüber hinaus dann zu, wenn eine Abbildung nicht, wie im Kontext eines Textes üblich, von einer Abbildungsunterschrift mit entsprechend detaillierterer Erklärung begleitet wird. Typisches Szenario wäre das Erstellen einer (ersten) Abbildung von Daten zum Vergleich mehrerer Datensätze für die weitere Auswertung und Diskussion.

Zunächst einmal sollten Sie fünf Funktionen definieren und darstellen und dabei darauf achten, dass die Wurzelfunktion und der natürliche Logarithmus nicht für negative x-Werte definiert sind. Entsprechend

sollten diese beiden Funktionen auch nur für das Intervall $x = [0, 3]$ dargestellt werden. Die entsprechende Definition der Funktionen und die zugehörigen `plot`-Befehle sind nachfolgend dargestellt.

Listing 9: Definition diverser Funktionen und Darstellung für unterschiedliche Invervalle

```
x2 = -3:0.01:3;
y4 = exp(x2);
y5 = x2.^2;
y6 = x2;

plot(x2,y4, x2,y5, x2,y6);

x3 = 0:0.01:3;
y7 = x3.^(1/2);
y8 = log(x3);

hold on;
plot(x3,y7,'m-')
plot(x3,y8,'k-')
hold off;
```

Mathematisch versiertere Personen können zwar schnell die dargestellten Funktionen erkennen und zuordnen, oft ist das aber nicht möglich, und hier hilft eine Legende, die die Linien (über die Farbe bzw. den Linienstil) entsprechend zuordnet, weiter. Der Befehl in MATLAB, um eine solche Legende zu erzeugen, lautet `legend`. In der einfachsten Form seiner Verwendung übergeben Sie dem Befehl lediglich eine Liste von Zeichenketten für die Anzahl der dargestellten Linien. Sie können entweder eine durch Kommata getrennte Liste von Zeichenketten direkt in die runden Klammern hinter dem Befehl `legend` schreiben, oder diese Liste von Zeichenketten über geschweifte Klammern in einem `cell array` anordnen. Letztere Variante erlaubt Ihnen auch, bequem außerhalb des `legend`-Befehls die Legende zu erzeugen (und ggf. zu generieren).

Darüber hinaus bietet Ihnen MATLAB auch noch die Möglichkeit, die Positionierung der Legende innerhalb gewisser Grenzen vorzugeben. Auch hier werden wieder Schlüssel-Wert-Zuweisungen verwendet, das zugehörige Schlüsselwort heißt `Location` (und nicht etwa `Position`). Mögliche Werte sind die vier Himmelsrichtungen und ihre Zusammensetzungen sowie `Best`, was MATLAB überlässt, die Legende so zu positionieren, dass am Wenigsten Überlapp mit den Linien in der Abbildung entsteht. Für Details sollten Sie die Hilfe zum Befehl `legend` konsultieren, z.B. über Eingabe von `doc legend`.

Zwei (einfache) Möglichkeiten, eine Legende in MATLAB zu definieren, sind Ihnen in nachfolgendem Listing angegeben, die zweite Variante kam für die Darstellung in Abb. 3 zum Einsatz. Beachten Sie, dass im ersten Fall die Beschriftungen als Liste von Zeichenketten dem Befehl `legend` direkt als Argumente übergeben wurden, während in zweitem Fall ein `cell array` verwendet wurde.

Listing 10: Definition der Legende

```
legend('Exponentialfunktion','Quadratische Funktion',...
    'Lineare Funktion','Wurzelfunktion','natuerlicher Logarithmus')

legend({'f(x) = exp(x)','f(x) = x^2','f(x) = x',...
    'f(x) = sqrt(x)','f(x) = ln(x)'},'Location','nw');
```

Sie sehen an diesem Beispiel auch schnell, wo die Grenzen von MATLAB sind, was die Interpretation von LaTeX-Steuerbefehlen für den Textsatz angeht. Während Sie eine Zahl über das Symbol ^ hochstellen und

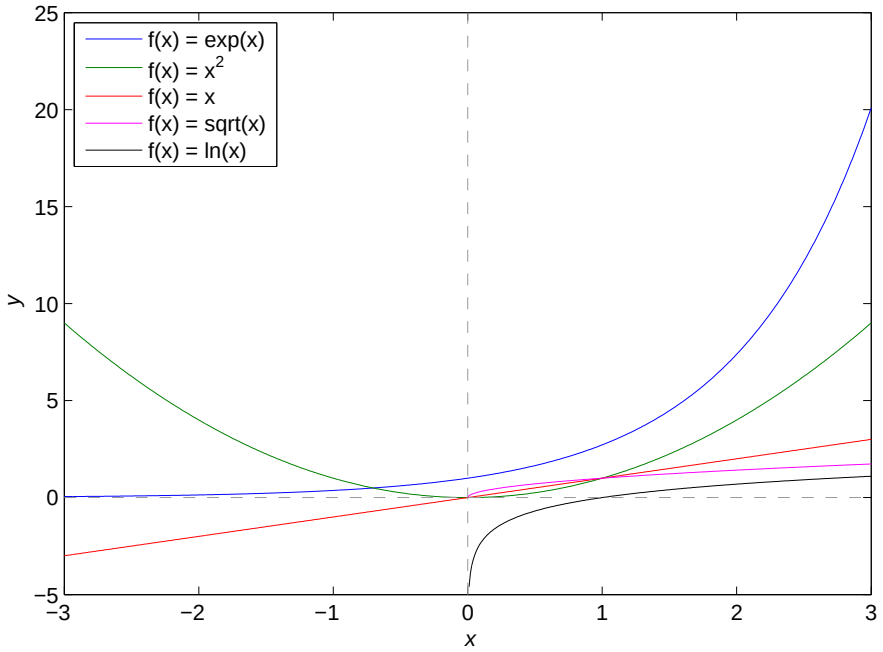

Abbildung 3: Grafische Auftragung mehrerer Funktionen und Zuordnung mittels Legende. Den MATLAB-Quellcode zur Darstellung der Funktionen finden Sie in Listing 9, den für die Legende in Listing 10. Beachten Sie, dass zusätzlich noch horizontale und vertikale gestichelte graue Linien durch den Koordinatenursprung eingezeichnet wurden, um die Verständlichkeit der Abbildung zu erhöhen.

ggf. über _ tiefstellen können, versteht MATLAB Befehle wie `\sqrt` zur Darstellung des Wurzelzeichens schon nicht mehr. An dieser Stelle haben andere Programme definitiv mehr zu bieten. Beachten Sie, dass Sie strenggenommen, wenn Sie die Formeln in der Legende angegeben, alle x und alle $f(x)$ kursiv setzen müssten, Zahlen und Operatoren (exp, sqrt, ln) sowie Klammern hingegen aufrecht.

Aufgabe 6—5 (Weitere Plotbefehle und Darstellungen)

Natürlich beherrscht MATLAB über den einfachen `plot`-Befehl hinaus noch eine ganze Reihe weiterer Darstellungsformen in zwei und drei Dimensionen. Eine Übersicht finden Sie auf der Webseite zu MATLAB[1]. Diese Seite hat darüber hinaus den Vorteil, dass Sie nicht nur eine Galerie an diversen Abbildungstypen vorfinden, sondern sich zur jeweiligen Abbildung auch den zugrundeliegenden Quellcode anschauen können.

Das Ihnen in dieser Aufgabe gegebene Beispiel einer Kugel, auf die Sie Punkte unterschiedlicher Größe einzeichnen sollen, hat einen realen Hintergrund in der Physikalischen Chemie. Wenn Sie numerische Simulationen orientierungsabhängiger Größen für eine ungeordnete Probe (z.B. in Lösung) durchführen wollen, dann müssen Sie über alle möglichen Orientierungen Ihrer Probe mitteln. Dieser Prozess wird meist als „Pulvermittelung“ bezeichnet. Konkret nehmen Sie dafür die vom ersten Aufgabenblatt bereits bekannte Euler-Drehmatrix und drehen schrittweise das Ihre Simulation beschreibende System. Idealerweise wollen Sie dabei eine Kugeloberfläche gleichmäßig abrastern, was allerdings kein triviales Unterfangen ist. Um sich diverse Möglichkeiten der Pulvermittelung zu visualisieren, kann es ganz praktisch sein, die einzelnen Orientierungen als Punkte auf einer Kugel darzustellen und ggf. mit ihrem Gewichtungsfaktor, der auch in der Mittelung verwendet wird, zu skalieren. Darüber hinaus sehen Sie, dass MATLAB Sie bei der Verwendung von Kugelkoordinaten unterstützt und Ihnen mitunter manche Denkarbeit bei der Transformation zwischen kartesischen und Kugelkoordinaten abnimmt.

[1] `http://de.mathworks.com/products/matlab/plot-gallery.html`

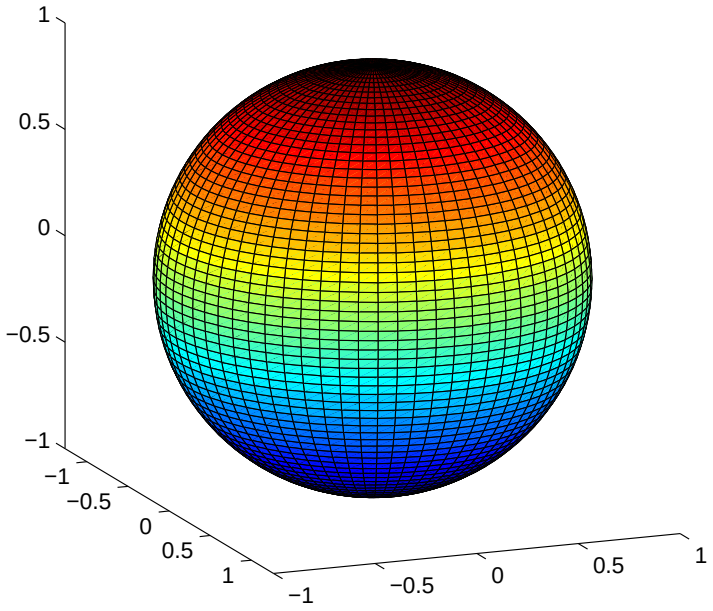

Abbildung 4: Grafische Darstellung einer Kugel in MATLAB. Den zugehörigen Quellcode, um zunächst die Koordinaten der Kugel in kartesischen Koordinaten zu erzeugen und anschließend als Oberfläche darzustellen, finden Sie in Listings 11 und 12. Wichtig ist die Einstellung des Seitenverhältnisses der Achsen über `axis equal`.

Eine einfache Kugel können Sie in MATLAB mit dem Befehl `sphere` erstellen, der Ihnen drei Matrizen mit den kartesischen Koordinaten zur Darstellung einer Kugel im dreidimensionalen kartesischen Koordinatensystem zurückgibt. Der entsprechende Aufruf in MATLAB sieht wie folgt aus:

Listing 11: Definition der Koordinaten zur Darstellung einer Kugel im kartesischen Koordinatensystem

```
[a,b,c] = sphere(70);
```

Der übergebene Parameter definiert so etwas wie die Gitterpunkte pro kartesischer Achse. Das Ergebnis, das Sie erhalten, ist natürlich nicht im eigentlichen Sinne eine Kugel, sondern ein geometrisches Gebilde, das eine Kugel durch einzelne Flächen annähert. Um diese Kugel dann grafisch als Oberfläche darzustellen, können Sie den Befehl `surface` verwenden:

Listing 12: Darstellung einer Kugel im kartesischen Koordinatensystem

```
surface(a,b,c);
axis equal;
view([68 14]);
```

Wenn Sie diese Zeilen in MATLAB ausführen, werden Sie als Resultat eine recht farbenfrohe Kugel erhalten. Das Ergebnis sehen Sie in Abb. 4. Die zweite Zeile des obigen Listings 12, `axis equal`, setzt das Seitenverhältnis der drei Achsen auf identische Werte, so dass die Kugel Ihnen auch als Kugel erkennbar wird und nicht aussieht wie ein Rotationsellipsoid. Die dritte Zeile, `view([68 14])`, setzt die Ansicht noch so, dass Sie schräg auf einen Pol der Kugel blicken und so ebenfalls einen besseren Eindruck der Kugel bekommen. Sie können selbstverständlich auch mit der Maus nach der Auswahl der richtigen Schaltfläche in der Symbolleiste oben am Rand des MATLAB-Abbildungsfensters die Ansicht interaktiv verändern.

Auch wenn farbenfrohe Darstellungen sicherlich Vorteile haben, eignet sich diese Darstellung der Kugel nur bedingt dafür, auf der Kugeloberfläche Punkte unterschiedlichen Durchmessers einzuzeichnen. Wollen Sie die Kugel lediglich als „Projektionsfläche“ für die eigentlichen Informationen verwenden, bevorzugen Sie vermutlich etwas gedecktere Farben. Eine Möglichkeit, das zu erreichen, ist wieder über Schlüssel-Wert-Paare, die Sie dem `surface`-Befehl mitgeben können:

Listing 13: Darstellung einer grauen Kugel im kartesischen Koordinatensystem

```
lightGrey = 0.6*[1 1 1];
surface(a,b,c,'FaceColor',lightGrey,'facealpha',0.3,...
    'EdgeColor',lightGrey,'edgealpha',0.3)
```

Hier sehen Sie einmal ein Beispiel, wie Sie sich selbst Farben als RGB-Vektor definieren und anschließend einsetzen können. Die beiden Angaben `facealpha` und `edgealpha` sorgen für eine transparente Darstellung der Kugel. Beachten Sie, dass Sie, sobald Sie eine Farbe für die Oberfläche festlegen, beim Export als PDF-Datei in der Regel keine Vektorgrafik mehr bekommen. Der Grund ist nicht ganz klar, hier zeigt sich eine der großen Schwächen von MATLAB in der Handhabung dreidimensionaler Darstellungen und Oberflächen. Andere Programme sind an dieser Stelle weitaus brauchbarer.

Um nun wie gefordert Punkte auf der Kugeloberfläche einzuzeichnen, können Sie die Funktion `scatter3` verwenden. `scatter3(X,Y,Z)` zeichnet Kreise an den durch die Vektoren `X`, `Y` und `Z` angegebenen Orten. Um diese Kreise auf die Kugeloberfläche zu setzen, müssen Sie sich also der trigonometrischen Funktionen bedienen. Gefragt war nach Punkten, die vom Nordpol der Kugel bis zum Äquator reichen und auf diesem Weg an Größe zunehmen.

Listing 14: Erzeugung der Koordinaten für die mit `scatter3` auf die Kugeloberfläche zu zeichnenden Kreise

```
theta = 0:pi/40:pi/2;

x = cos(theta);
y = zeros(1,length(theta));
z = sin(theta);

area = 100*cos(theta);
```

Wie Sie sehen, wird zunächst ein Vektor `theta` mit Werten zwischen 0 und $\frac{\pi}{2}$ definiert und dann über die trigonometrischen Funktionen die Koordinaten (x, y, z) der Positionen auf der Kugeloberfläche berechnet. Der Vektor `area` enthält die mit $\cos\theta$ gewichtete und mit dem Faktor 100 skalierte Größe der zu zeichnenden Punkte. Diese Größe können Sie dem Befehl `scatter3` als viertes, optionales Argument beim Aufruf übergeben.

Der eigentliche Aufruf von `scatter3` könnte wie im nachfolgenden Listing dargestellt aussehen. Der Grund, warum dem Vektor mit der Größe der Punkte für jeden Punkt der Wert 1 hinzuaddiert wird, liegt darin, dass ansonsten der Kreis am Pol eine verschwindende Ausdehnung hätte ($\cos(\frac{\pi}{2}) = 0$). Die Angabe der Schlüssel-Wert-Paare für die Farbe der Kanten und der Füllung der Kreise über `MarkerEdgeColor` und `MarkerFaceColor` dient lediglich der Ästhetik.

Listing 15: Einzeichnung der Kreise auf der Kugeloberfläche mit `scatter3`

```
hold on;
scatter3(x,y,z,(area+1),'MarkerEdgeColor','k','MarkerFaceColor',[0 .75 .75]);
hold off;
```

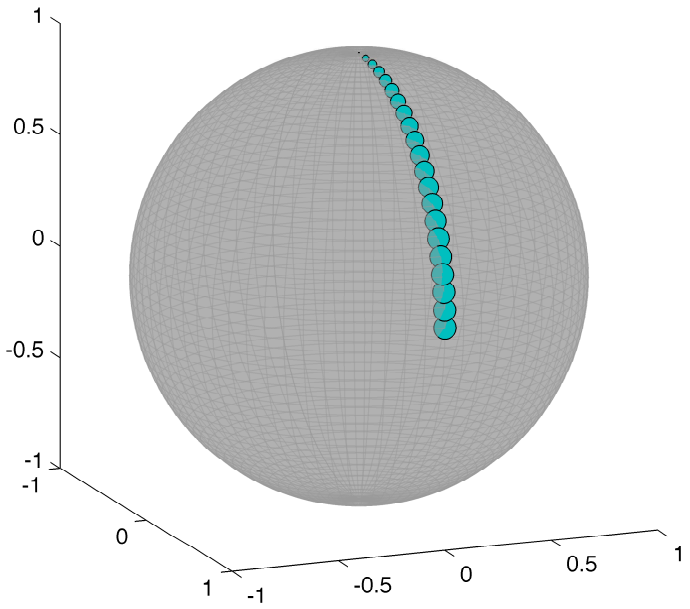

Abbildung 5: Grafische Auftragung einer Kugel mit Punkten auf der Kugeloberfläche. Diese Abbildung ist ein gutes Beispiel dafür, wie schwer sich MATLAB mitunter mit einem sinnvollen Export solcher dreidimensionaler Darstellungen tut. Auch wenn Sie diese Abbildung als PDF-Datei exportieren, ist das Ergebnis keineswegs eine vektorisierte Grafik, sondern eine Rastergrafik. Abhilfe beim Export schafft hier ggf. nur das Herumspielen mit diversen Parametern inkl. der Option für die Auflösung. Gegenüber der ursprünglichen Darstellung wurde die Kugel etwas gedreht, um die Anordnung der Punkte besser zu visualisieren.

Das finale Ergebnis dieser Darstellung sehen Sie in Abb. 5. Beachten Sie, dass diese Abbildung trotz Exports als PDF-Datei eine Rastergrafik statt einer Vektorgrafik ist. Mir ist aktuell kein Weg bekannt, dieses Problem zu lösen. Es ist aber gut möglich, dass es in aktuelleren MATLAB-Versionen behoben wurde. Mit MATLAB 2014b wurden die Grafikfunktionen vollständig überarbeitet, was mitunter zu einem komplett anderen Verhalten von Funktionen führt und ein Grund war, sich im Kurs noch auf die etwas ältere Version MATLAB 2014a zu beschränken.

Haben Sie an diesen Stellen immer im Blick, dass es nicht besonders hilfreich ist, wenn Sie Programme schreiben, die jeweils nur unter den neuesten Versionen einer kommerziellen Sprache wie MATLAB lauffähig sind. Dadurch schränken Sie die Weiterverwendbarkeit Ihrer Programme unnötig ein, zumal Sie nicht davon ausgehen dürfen, dass jeder potentielle Nutzer immer Zugriff auf die jeweils neueste Version von MATLAB hat.

Aufgabe 6—6 (Skalierte Darstellung von Spektren)

Nach diesem Ausflug in die Darstellung dreidimensionaler Körper sollten Sie in dieser letzten Aufgabe mehrere UV/vis-Spektren in einer Abbildung darstellen. Der Fokus liegt hierbei darauf, möglichst sinnvolle und aussagekräftige Abbildungen zu erstellen, die die wesentlichen Informationen, die Sie aus den Messdaten ziehen können, für den Betrachter unmittelbar zugänglich machen. Darüber hinaus können Sie hier die im Rahmen des letzten Aufgabenblattes entwickelte Funktion `loadUVVisData` gewinnbringend einsetzen, um die Daten einzulesen.

Es handelt sich um vier unterschiedliche UV/vis-Spektren, und zwar von einem Polymer (PCDTBT) und drei seiner Bausteine (TBT, CbzTBT und CbzTBTCbz). Hier interessiert uns nicht die chemische

Struktur dieser Substanzen im Detail, sondern nur, dass die Kettenlänge der Moleküle in dieser Reihenfolge zunimmt: TBT < CbzTBT < CbzTBTCbz < PCDTBT. Da es sich um konjugierte Moleküle handelt, erwarten Sie entsprechend eine Rotverschiebung der Absorption mit zunehmender Kettenlänge.

Dank Ihrer selbstgeschriebenen Einleseroutine für UV/vis-Daten im hier vorliegenden Format im Rahmen des vorangegangenen Aufgabenblattes stellt sich der Import der Daten in MATLAB sehr einfach dar. Das Ergebnis der vier nachfolgend gezeigten Aufrufe ist – zumindest wenn Sie die Routine hinsichtlich ihrer Schnittstelle so implementiert haben wie in der Lösung zum letzten Aufgabenblatt vorgestellt – jeweils eine Matrix mit zwei Spalten, in der ersten Spalte die Wellenlängenachse, in der zweiten die Intensitätswerte.

Listing 16: Einlesen der vier UV/vis-Spektren mit der Routine `loadUVVisData`

```
TBT       = loadUVVisData('TBT.txt');
CbzTBT    = loadUVVisData('CbzTBT.txt');
CbzTBTCbz = loadUVVisData('CbzTBTCbz.txt');
PCDTBT    = loadUVVisData('PCDTBT.txt');
```

Eine erste grafische Darstellung ist ebenfalls einfach über den `plot`-Befehl möglich. Allerdings werden Sie schnell feststellen, dass zumindest eines der vier Spektren gegen kurze Wellenlängen so stark absorbierte, dass der Detektor in die Sättigung gefahren wurde, was sich am „Rauschen“[2] in diesem Bereich des Spektrums erkennen lässt. Abhilfe schafft hier die Beschränkung der Wellenlängenachse auf das Intervall von 300–800 nm. Sie können das ganz bequem dadurch erreichen, dass Sie nachträglich die Eigenschaft `XLim` der Achse setzen. Das sorgt gleichzeitig für eine Reskalierung der *y*-Achse – eine der Stärken von MATLAB, wenn es Ihnen darum geht, schnell halbwegs ansehnliche Abbildungen zu bekommen, gleichzeitig aber mitunter auch nervig, wenn Sie mühsam (und bewusst) Achsenbereiche händisch festgelegt haben.[3] Die Befehle zum Setzen der Achsenbeschriftungen und der Legende sind Ihnen aus vorangegangenen Aufgaben bekannt. Das Ausweichen auf englische Achsenbeschriftungen ist hier der Tatsache geschuldet, dass man so Probleme mit der Darstellung von Umlauten umgehen kann.

Listing 17: Erste Darstellung der vier UV/vis-Spektren

```
plot(...
    TBT(:,1),TBT(:,2),...
    CbzTBT(:,1),CbzTBT(:,2),...
    CbzTBTCbz(:,1),CbzTBTCbz(:,2),...
    PCDTBT(:,1),PCDTBT(:,2)...
    );

set(gca,'XLim',[300 800]);
xlabel('{\it wavelength} / nm');
ylabel('{\it absorbance} / a.u.');

legend({'TBT','CbzTBT','CbzTBTCbz','PCDTBT'});
```

Das Ergebnis der oben angegebenen MATLAB-Befehle ist in Abb. 6 wiedergegeben. Für einen ersten Eindruck ist diese Abbildung sicherlich schon ganz gut geeignet, allerdings ist es aufgrund der drama-

[2] Rauschen ist hier nicht der korrekte Begriff. Vermutlich handelt es sich um stark nichtlineares Verhalten des Detektors. Auf jeden Fall sehen Sie hier sehr anschaulich, warum man mit normalen Absorptionsspektrometern tunlichst nicht bei Absorbanzen deutlich über eins messen sollte.

[3] Es ist weder intuitiv noch notwendigerweise sinnvoll, dass ein Befehl zur Reskalierung der *x*-Achse dramatischen Einfluss auf die Darstellung der *y*-Achse haben soll. Sie können dieses Verhalten für eine Abbildung in MATLAB auch abstellen, indem Sie den Wert einer entsprechenden Eigenschaft des Abbildungsfensters bzw. der Achse umsetzen. Für Details sollten Sie die MATLAB-Dokumentation konsultieren.

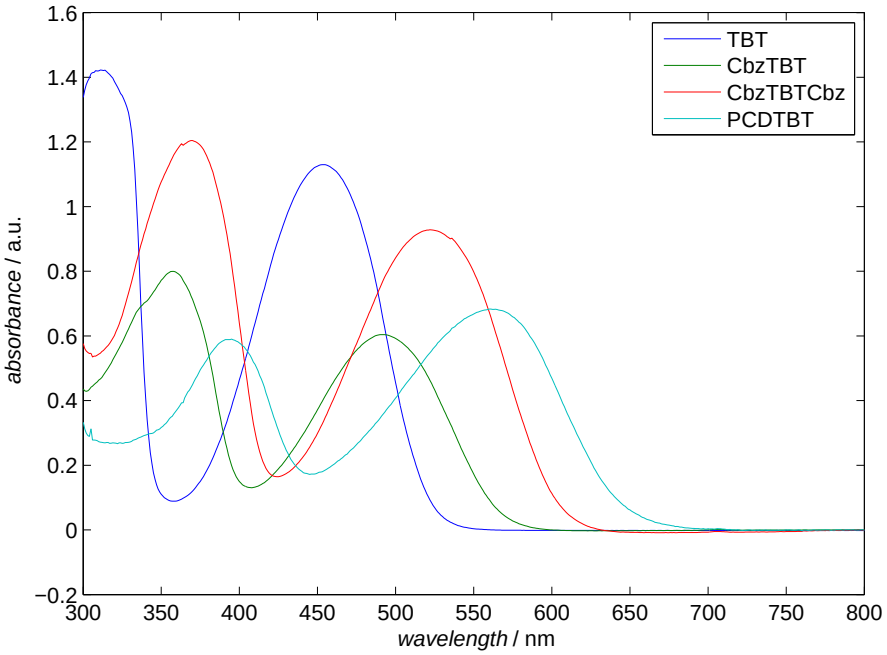

Abbildung 6: UV/vis-Spektren des Polymers PCDTBT und seiner Bausteine. Die zur Erzeugung dieser Darstellung verwendeten MATLAB-Befehle finden Sie in Listing 17. Es handelt sich hierbei weitestgehend um Standard-Befehle. Lediglich die x-Achse wurde reskaliert und bei 300 nm zu kürzeren Wellenlängen hin abgeschnitten.

tisch unterschiedlichen Intensitäten der einzelnen Absorptionsbanden nicht einfach, die angesprochene erwartete Rotverschiebung mit zunehmender Kettenlänge eindeutig zu erkennen. Hierfür wäre es weitaus besser, die Spektren auf gleiches Maximum der Intensität der jeweils längstwelligen Absorptionsbande zu normieren. Wenn Sie die Spektren in Abb. 6 daraufhin noch einmal genauer anschauen, werden Sie feststellen, dass Sie mit dem Befehl `max`, der Ihnen das Maximum eines Vektors zurückgibt, nicht besonders weit kommen, da für alle Substanzen mit Ausnahme des Polymers die Absorptionsbande, die blauverschoben von der längstwelligen Absorptionsbande auftritt, in ihrer Intensität stärker als die längerwellige Absorptionsbande ist.

Im vorliegenden Beispiel kommt Ihnen eine Eigenschaft der vier dargestellten Spektren zugute: Wenn Sie jeweils nur die Intensität zwischen 400 und 800 nm Wellenlänge betrachten, sehen Sie, dass das Maximum der Intensität in diesem Bereich tatsächlich die Absorptionsbande ist, die am weitesten im langwelligen Bereich liegt. Es handelt sich hierbei, wie in der Aufgabenstellung angesprochen, um eine sogenannte „CT-Bande“. Um nun diese CT-Bande für alle vier Spektren auf gleiches Maximum zu skalieren, können Sie das Maximum für den Wellenlängenbereich von 400–800 nm bestimmen und das gesamte Spektrum durch diesen Wert teilen. So normieren sie das Maximum der CT-Bande auf eine Absorbanz von eins. In MATLAB können Sie sich für diese Aufgabe die sogenannte „logische Indizierung“ zunutze machen. Ein Beispiel ist Ihnen nachfolgend gezeigt.

Listing 18: Beispiel logischer Indizierung, um (nur) die CT-Bande auf eins zu normieren

```
TBT(:,2)=TBT(:,2)/max(TBT(TBT(:,1)>=400,2));
```

Diese kompakte Zeile ist mit Sicherheit auf den ersten Blick etwas gewöhnungsbedürftig. Deshalb eine detaillierte Erklärung. Sie weisen der zweiten Spalte der Matrix `TBT`, die die UV/vis-Daten des Moleküls TBT enthält, eine intensitätsnormierte Version dieser Daten zu. Dazu teilen Sie die Intensitätswerte durch das Maximum der Intensitätswerte im Bereich $\lambda \geq 400$ nm. Der Befehl `max` macht genau das, was

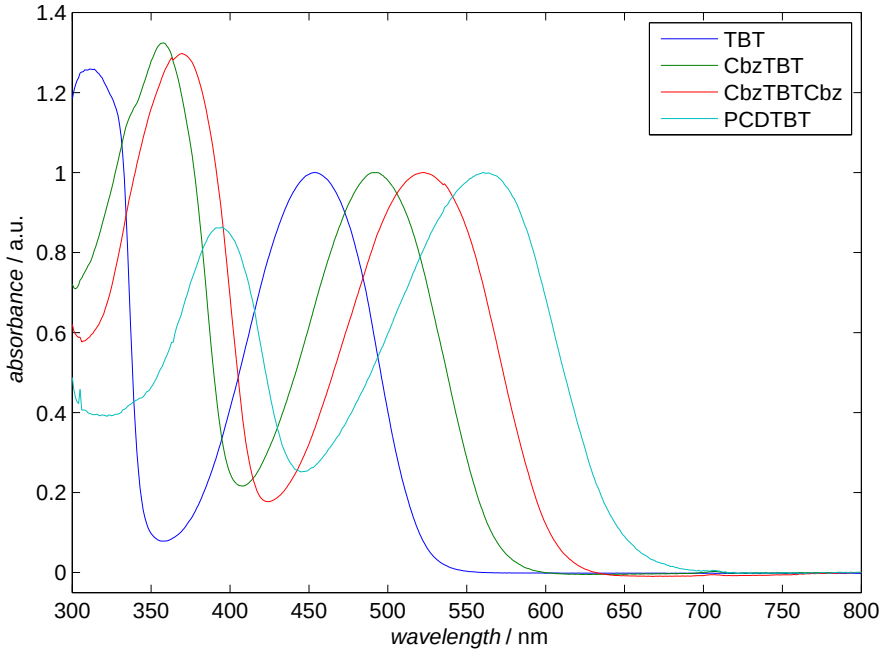

Abbildung 7: UV/vis-Spektren des Polymers PCDTBT und seiner Bausteine, normiert auf die CT-Bande. Aus dieser Abbildung ist sofort eine zentrale Aussage dieser Spektren erkennbar: Die CT-Bande verschiebt sich mit zunehmender Kettenlänge zu größeren Wellenlängen, was auf eine Zunahme der Größe des konjugierten Systems und damit auf die Delokalisierung der Elektronen hindeutet. Die wesentlichen zur Erzeugung dieser Darstellung verwendeten MATLAB-Befehle finden Sie in Listing 20. Vgl. auch Listings 19 und 17.

man von ihm erwartet, nämlich den größten Wert eines Vektors zurückzugeben. Hier wenden Sie `max` auf die zweite Spalte von `TBT` an, allerdings nur auf den Bereich `TBT(:,1)>=400` Da Sie wissen, dass die erste Spalte der Matrix `TBT` die Wellenlängenwerte enthält und Sie nur an jenen Intensitätswerten des Spektrums interessiert sind, die im Bereich $\lambda \geq 400$ nm liegen, können Sie das logisch abfragen. Der Ausdruck `TBT(:,1)>=400` liefert Ihnen einen Vektor mit Booleschen Werten von der Länge von `TBT(:,1)` zurück, den Sie dann wieder als Index für `TBT(:,2)` einsetzen können (und das hier auch tun). Natürlich können Sie sich, zumal Sie diese Operation auf alle vier Datensätze anwenden wollen, diesen Ausdruck in eine anonyme Funktion auslagern:

Listing 19: Anonyme Funktion zur Normierung des Maximums der CT-Bande auf eins

```
normaliseCTband = @(x)x(:,2)/max(x(x(:,1)>=400,2));
```

Damit wird die geforderte Darstellung der Spektren mit normierter CT-Bande geradezu zum „Kinderspiel“, wie Ihnen nachfolgendes Listing zeigt. Beachten Sie, dass in diesem Fall der Bereich der y-Achse ebenfalls etwas angepasst wurde, zumal Sie nach „unten“ nicht so viel Platz brauchen, wie MATLAB automatisch einräumt. Der Befehl dazu ist analog zum oben gezeigten für die Skalierung der x-Achse.

Listing 20: Darstellung der vier UV/vis-Spektren mit normierter CT-Bande

```
plot(...
    TBT(:,1),       normaliseCTband(TBT),...
    CbzTBT(:,1),    normaliseCTband(CbzTBT),...
    CbzTBTCbz(:,1),normaliseCTband(CbzTBTCbz),...
    PCDTBT(:,1),    normaliseCTband(PCDTBT)...
    );
```

Das Ergebnis dieses `plot`-Befehls ist in Abb. 7 dargestellt. Hier können Sie auf einen Blick erkennen, dass sich die CT-Bande mit zunehmender Kettenlänge des betrachteten Moleküls zu größeren Wellenlängen hin verschiebt.

Natürlich gäbe es auch noch eine weitere Möglichkeit, diese Aussage noch offensichtlicher darzustellen: wenn Sie nur noch die Wellenlänge der CT-Bande für jedes Molekül gegen das Molekül auftragen. Dazu müssen Sie zunächst aus den vier Datensätzen jeweils die Wellenlängenposition der CT-Bande extrahieren. Hier kommt Ihnen eine alternative Möglichkeit des Aufrufes der Funktion `max` zugute (gleiches gilt für die korrespondierende Funktion `min`): Wenn Sie diese Funktion(en) mit zwei statt einem Rückgabeparameter aufrufen, bekommen Sie als zweiten Parameter die Position des Maximums (Minimums) im übergebenen Datenvektor. Diese Information können Sie nun wiederum dazu nutzen, aus dem Vektor mit der Wellenlängeninformation die zugehörige Wellenlänge zu gewinnen. Worauf Sie jetzt noch achten müssen, ist die Tatsache, dass Ihnen dieser Index, wenn Sie das Maximum nur für einen bestimmten Bereich des Vektors durchsuchen, um ein Offset verschoben ist. Dieses Offset müssen Sie also ebenfalls noch herausbekommen, können dazu allerdings ebenfalls wieder logische Indizierung mit umgekehrter Logik (der Index des Maximums des Wellenlängenvektors für $\lambda < 400$ nm) verwenden.

Gehen wir nun noch davon aus, dass Sie nicht sicher sein können, dass alle vier Spektren über den gleichen Wellenlängenbereich gemessen wurden. Das ist in der Praxis deutlich häufiger der Fall, als Sie vielleicht glauben wollen – spätestens dann, wenn Sie die Daten mehrerer unterschiedlicher Messtage vergleichen wollen, taucht dieses „Problem“ mit hübscher Regelmäßigkeit auf. Sie wollen also Quellcode schreiben, der all diese Unwägbarkeiten elegant umschifft. Eine Möglichkeit, die von verschiedenen bislang vorgestellten Konzepten Gebrauch macht, ist nachfolgend wiedergegeben:

Listing 21: Extraktion der Wellenlänge des Maximums der CT-Bande aus den Daten

```
spectra = {TBT CbzTBT CbzTBTCbz PCDTBT};
lambdaCTmax = zeros(length(spectra),1);
for spectrum = 1:length(spectra)
    [~,offset] = max(spectra{spectrum}(spectra{spectrum}(:,1)<400,1));
    [~,CTmax] =  max(spectra{spectrum}(spectra{spectrum}(:,1)>=400,2));
    lambdaCTmax(spectrum) = spectra{spectrum}(CTmax+offset,1);
end
```

Zunächst einmal wollen Sie sich Arbeit sparen und deshalb über alle vier Datensätze in einer `for`-Schleife iterieren und jeweils nur die Position des Maximums der CT-Bande (in nm) erhalten. Da es sich bei den Daten um Matrizen handelt, von denen Sie nicht sicher sein können, dass sie immer identische Dimensionen haben, ist ein `cell array` eine einfache Möglichkeit, die sich zudem noch relativ einfach indizieren lässt. Das ist das, was in der ersten Zeile passiert: `spectra` ist ein `cell array` mit den vier Datensätzen als Elemente. Anschließend erzeugen Sie noch den Vektor, der die Wellenlängenmaxima der CT-Bande der vier Substanzen enthalten soll, in der richtigen Größe, damit sich seine Größe in der anschließenden Schleife nicht ändert (und der MATLAB-Editor keine diesbezüglichen Warnungen ausgibt).

In der `for`-Schleife werden nun die Indices des Wellenlängenvektors für das Offset ($\lambda < 400$ nm) und für das Maximum der CT-Bande (abzüglich des zuvor berechneten Offsets) durch logische Indizierung aus dem jeweiligen Datensatz herausgeholt. Hier kommt jeweils der zweite Rückgabeparameter der Funktion `max` zum Einsatz. Da Sie hier kein Interesse am ersten Rückgabeparameter haben, aber andererseits auch nicht vom MATLAB-Editor mit Warnungen bedacht werden wollen, dass Sie Variablen definieren, die Sie hernach nicht mehr verwenden, können Sie das erste Rückgabeargument durch Eingabe der Tilde ignorieren. Das ist ein recht praktisches universell einsetzbares Konzept für Rückgabeparameter. Für Übergabeparameter von Funktionen ist es hingegen (leider) nicht einsetzbar. Nachdem Sie nun sowohl das

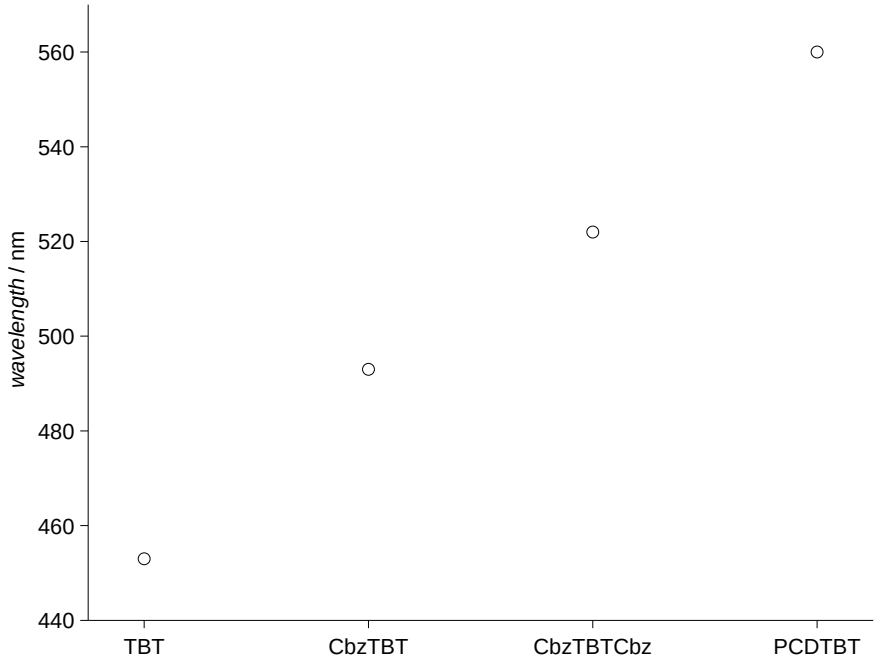

Abbildung 8: Auftragung der Wellenlänge des Maximums der CT-Bande gegen die jeweilige Substanz. Diese Abbildung ist auf die zentrale Aussage der vorangegangenen skalierten Darstellung der Spektren (Abb. 7) reduziert: Die CT-Bande verschiebt sich mit zunehmender Kettenlänge zu größeren Wellenlängen. Die zur Erzeugung dieser Darstellung verwendeten MATLAB-Befehle finden Sie in Listing 22.

Offset als auch den Offset-behafteten Index des Wellenlängenmaximums der CT-Bande in Händen halten, können Sie mit dieser Information die Wellenlänge des Maximums der CT-Bande aus dem Datensatz extrahieren.

Eine weitere Besonderheit des oben angegebenen Quellcodes ist die doppelte Indizierung, die Sie benötigen, um auf die Datensätze im `cell array` zuzugreifen. Beachten Sie, dass Sie den ersten Index *zwingend* in geschweifte Klammern setzen müssen, da Sie nur so an den `Inhalt` dieses Elementes des `cell arrays` herankommen, an dem Sie ja interessiert sind, nämlich die Matrix mit den Daten Ihres zugehörigen Datensatzes. Die Indices in runden Klammern, die unmittelbar auf den Index in geschweiften Klammern (für das `cell array`) folgen, sind wieder identisch mit dem bereits weiter oben angegebenen Quellcode. Im Endergebnis enthält der Vektor `lambdaCTmax` die Wellenlängenwerte für das Maximum der CT-Bande der vier Datensätze. Diesen Vektor gilt es nun noch gegen die Substanzen aufzutragen und die Achsen entsprechend zu beschriften.

Der komplette Quellcode ist Ihnen nachfolgend wiedergegeben, das Ergebnis dieser Befehle sehen Sie in Abb. 8. Eine detaillierte Erklärung der einzelnen Zeilen folgt auf das Listing.

Listing 22: Auftragung der Wellenlänge des Maximums der CT-Bande gegen die Substanzen

```
plot(lambdaCTmax,'ko');
ylabel('{\it wavelength} / nm');
set(gca,'XLim',[.75 4.25],'YLim',[440 570]);

substances = {'TBT','CbzTBT','CbzTBTCbz','PCDTBT'};
set(gca,'XTick',(1:4),'XTickLabel',substances);
set(gca,'TickDir','out','Box','off');
```

Die eigentliche Darstellung der Wellenlänge der Maxima der CT-Bande ist denkbar einfach, zumal Sie hier

einfach die Werte gegen den Index des Vektors auftragen können, also effektiv den `plot`-Befehl mit nur einem Argument aufrufen. Da Sie an dieser Stelle die Datenpunkte allerdings *unter keinen Umständen* miteinander verbinden sollten, müssten Sie dem `plot`-Befehl noch mindestens ein optionales Argument für die Symbole mitgeben. Anschließend können Sie hinsichtlich der Achsenskalierung noch etwas Kosmetik betreiben, zumal Sie nicht wollen, dass der ganz linke Datenpunkt auf der y-Achse sitzt. Ob Sie hierfür aufeinander folgende Aufrufe von `xlim` und `ylim` verwenden oder das in ein oder zwei Aufrufe von `set` verpacken, ist eine Frage persönlicher Vorlieben.

Anschließend müssen Sie nur noch die x-Achse entsprechend beschriften. Dazu werden zunächst die Positionen der Skalenstriche (*tick marks*) über die Eigenschaft `XTick` auf die Positionen 1–4 reduziert und anschließend die zugehörigen Beschriftungen über die Eigenschaft `XTickLabel` gesetzt. Dieser Eigenschaft können Sie als Wert ein `cell array` mit Zeichenketten übergeben – in diesem Fall die Liste der Substanznamen in `substances`. Die letzte Zeile setzt aus rein kosmetischen Gründen noch zwei weitere Eigenschaften der Achsen: `TickDir` gibt an, ob die Skalenstriche innen oder außen an den Achsen erscheinen, `Box` definiert, ob auch auf den den Achsen jeweils entgegengesetzten Seiten Skalenstriche (und ein den gesamten Graphen einhüllender Kasten) erscheinen.

Noch eine letzte Anmerkung zur Darstellung der Daten in Abb. 8: Der fast lineare Verlauf der Datenpunkte, also die lineare Rotverschiebung der CT-Bande mit zunehmender Kettenlänge der Substanz, lässt keine Rückschlüsse auf den quantitativen Zusammenhang zwischen Kettenlänge und Absorptionsmaximum zu. Dazu müssten Sie auf der x-Achse statt der Substanzen (in äquidistanten Schritten) die realen Kettenlängen der beteiligten Substanzen auftragen. Trotzdem könnte man geneigt sein, eine (zarte) Linie durch die Datenpunkte zu zeichnen – quasi als Hilfe für das Auge des Betrachters. Hier käme am Ehesten eine Ausgleichsgerade durch alle vier Datenpunkte in Betracht, auch wenn der hier dargestellte Zusammenhang explizit keine physikalische Interpretation der Parameter einer solchen linearen Regression erlaubt. Die für die Erstellung einer solchen Ausgleichsgerade notwendigen Werkzeuge in MATLAB lernen Sie im Rahmen des nächsten Aufgabenblattes kennen.