Ü 4 Vierte Übungseinheit

Inhalt der vierten Übungseinheit:

- Eindeutig, nicht eindeutig und gar nicht lösbare Systeme
- Fehlerempfindlichkeit, Matrixnormen, Konditionszahl
- Determinante, Komplexität
- Modelle an Daten anpassen
- Lineare Datenmodelle

Eine Anleitung für diese Übungseinheit

Im aktuellenVorlesungsskript behandelt Kapitel 3 lineare Gleichungssysteme mit quadratischer Matrix (mit genausoviel Gleichungen wie Unbekannten). In den Folien zur 4. Vorlesung können sie dazu die Abschnitte

1. Lösbarkeit, Fehlerempfindlichkeit und
4. Direkte Verfahren

zur Wiederholung durchklicken.

Insbesondere die Skriptum-Kapitel 3.2, 3.4, 3.6, 3.7.1 und 3.8 bringen Material, das direkt mit den Übungaufgaben 31 – 35 zusammenhängt.

Die weiteren Aufgaben in dieser Übungseinheit befassen sich damit, wie sich aus Datensätzen die Parameter eines Modells schätzen lassen; zum Beispiel: aus der Anzahl von infizierten Personen die Parameter eines exponentiellen Wachstumsmodells, Anleitung dazu in den Übungsunterlagen Ü 4.4.2.

Dazu sollten Sie in den Vorlesungsfolien das Thema

5. Überbestimmte Systeme

durchklicken und Kapitel 5 im Vorlesungsskript durchblättern. Wir werden die Theorie noch ausführlicher behandeln, aber zum praktischen Einsatz reicht vorläufig, zu wissen:

- Ein System mit mehr Gleichungen als Unbekannten ist im Regelfall nicht exakt lösbar
- Eine Näherungslösung (die „am wenigsten falsche Lösung“) lässt sich mit der Methode der kleinsten Fehlerquadrate bestimmen.
- MATLABS Operator \ findet bei überbestimmten Gleichungssystemen die kleinste-Quadrate-Näherungslösung.

Ü 4.1 Gleichungssysteme, Lösungsmannigfaltigkeiten

Können Sie ein lineares Gleichungssystem mit zwei Gleichungen und zwei Unbekannten immer lösen?

> Die Faustregel „Bei genausoviel Gleichungen wie Unbekannten gibt es immer eine Lösung“ ist falsch. FALSCH! **FALSCH!!**

Bei den drei Gleichungssystemen

$$\begin{aligned} x + y &= 2 \\ 2x + 2y &= 4 \end{aligned} \qquad \begin{aligned} x + y &= 2 \\ 2x + 2y &= 3 \end{aligned} \qquad \begin{aligned} x + y &= 2 \\ x + 2y &= 3 \end{aligned}$$

sehen Sie hoffentlich mit freiem Auge: Beim ersten und beim dritten sind $x = 1,\ y = 1$ Lösungen. Das mittlere System ist unlösbar. Beim ersten System gibt es allerdings noch unendlich viele weitere Lösungen.

Wiederholen Sie, was Sie dazu in Mathematik 1 gelernt haben. Im Skriptum, Kapitel 3.4 finden Sie weitere Informationen. Sie sollten mit folgenden Themen vertraut sein:

- Matrix-Rang, Rang der erweiterten Matrix, MATLAB-Befehle `rank(A), rank([A,b])`, Fallunterscheidungen zur Lösbarkeit.
- Stufenform eines Gleichungssystems, MATLAB-Befehl `rref([A,b])`
- Allgemeine Lösung des homogenen Systems, Nullraum, MATLAB-Befehl `null(A)`
- Spezielle Lösung des inhomogenen Systems, MATLAB-Befehle `A\b` bei vollem Rang, `pinv(A)*b` bei lösbaren Systemen mit Rang kleiner Zeilenzahl.
- Bei nicht lösbaren Systemen liefert `pinv(A)*b` die „am wenigsten falsche“ Antwort (mit kleinstem Fehler in der 2-Norm)

Aufgabe 31: Gleichungssysteme, Lösungsmannigfaltigkeiten

Welche der folgenden Gleichungssysteme $A\mathbf{x} = \mathbf{b}$ sind eindeutig, mehrdeutig oder gar nicht lösbar? Geben Sie, wenn möglich, die (oder eine) Lösung an. Bei mehrdeutigen Lösungen: wie viele freie Parameter hat die Lösungsschar?

Weitere Erläuterungen, Arbeitsschritte
Die möglichen Fälle zur Lösbarkeit entscheiden Sie über Rang der Matrix und Rang der erweiterten Matrix, Befehle `rank(A), rank([A,b])`. Als Alternative können Sie auch `rref([A,b])` verwenden.

Wenn eine eindeutige Lösung existiert, ist `A\b` der richtige Befehl.

Wenn eine Lösungsschar existiert, liefert `pinv(A)*b` eine mögliche Lösung, und zwar jene mit kleinstem Absolutbetrag. Der Standard-Befehl `A\b` kann auch funktionieren; er liefert einen Lösungsvektor mit möglichst vielen Komponenten gleich 0.

Die vollständige Darstellung einer Lösungsmannigfaltigkeit können Sie aus dem Ergebnis von `rref([A,b])` ablesen. Das erfordert aber noch etwas Umformen.

Alternative: Die allgemeine Lösung besteht aus einer speziellen Lösung (bestimmt mit `pinv(A)*b`) plus einer Linearkombination der Spaltenvektoren des Nullraumes (bestimmt mit `null(A)`).

1. $$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

2. $$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

3.
$$A = \begin{bmatrix} 10 & 9 & 8 & 7 \\ 6 & 5 & 4 & 3 \\ 2 & 1 & 0 & -1 \\ -2 & -3 & -4 & -5 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

4.
$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Aufgabe 32: LR-Zerlegung

Das Skriptum behandelt in Kapitel 3 Gauß-Elimination, LR-Zerlegung, Vorwärts-Elimination, Rücksubstitution. Dazu listet es auch JAVA-Codes. Hier eine Portierung in MATLAB. Wenn Sie diese Aufgabe durcharbeiten, haben Sie Ihren ersten eigenen Gleichungslöser programmiert.

> (Warnung: Die Programme dienen zur Illustration der Grundform der Verfahren. Für den praktischen Einsatz sind sieso noch nicht geeignet, es wäre zu ergänzen: Pivotierung, Absicherungen gegen singuläre Matrizen, Division durch Null...

Verwenden Sie die Angabe des Rechenbeispiels im Skript, Abschnitt3.6: Gegeben sei das Gleichungssystem $A \cdot \mathbf{x} = \mathbf{b}$ mit

$$A = \begin{bmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{bmatrix} \quad , \quad \mathbf{b} = \begin{bmatrix} 6 \\ 6 \\ 14 \end{bmatrix}$$

Sie können damit alle Zwischenergebnisse nachvollziehen.

Tippen Sie nicht ab, das ist zu fehleranfällig. Verwenden Sie Copy-Paste aus der PDF-Datei!

Schritt 1: LR-Zerlegung. Die beiden Matrizen L und R werden aus Effizienz-Gründen nicht extra gespeichert, sondern ersetzen die A-Matrix. Die L-Matrix ohne die Eins-Diagonale besetzt das Dreieck unterhalb der Hauptdiagonale; Die R-Matrix befüllt das obere Dreieck.

```
%% Zerlegung A = LR
% geht schief, wenn Null in der Diagonale auftritt!
for k=1:n
    for i=k+1:n
        A(i,k)=A(i,k)/A(k,k);
        for j=k+1:n
            A(i,j) = A(i,j)-A(i,k)*A(k,j);
        end
    end
end
```

Die Zerlegung ist damit abgeschlossen, vergleichen Sie (und beachten Sie: die beiden Ausgabe-Matrizen überschreiben die entsprechenden Bereiche in A):

$$A \text{ (Eingabe)} \begin{bmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{bmatrix} \quad , \quad L \cdot R \text{ (Ausgabe)} \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 & 7 \\ 0 & 8 & 9 \\ 0 & 0 & 10 \end{bmatrix}$$

Erst jetzt kommt die rechte Seite b ins Spiel: Die Vorwärts-Elimination löst $Ly = b$ und berechnet so das Zwischenresultat y. Im Code wird y gleich als x gespeichert.

```
%% Vorwärts-Elimination Ly=b
x = b;
for i=1:n
    for j=1:i-1
        x(i) = x(i) - A(i,j)*x(j);
    end
end
```

Das Zwischenresultat y bzw. x entspricht der letzten Spalte nach Abschluss des Eliminationsverfahrens, vor der Rücksubstitution, vergleichen Sie im Skript:

$$[A\,\mathbf{b}]^{(2)} = \begin{bmatrix} 5 & 6 & 7 & 6 \\ 0 & 8 & 9 & -6 \\ 0 & 0 & 10 & 20 \end{bmatrix}$$

```
%% Rücksubstitution Rx=y
for i=n:-1:1
    for j=i+1:n
        x(i) = x(i)-A(i,j)*x(j);
    end
    x(i) = x(i)/A(i,i);
end
```

Nun ist x die Lösung des Systems.

Wie gut ist diese ganz naive Implementierung im Vergleich zu „richtigen“ Gleichungslöse-Programmen? Testen Sie dazu mit zufällig erzeugten Systemen

```
n=4;
A=floor(10*rand(n));
b=floor(10*rand(n,1));
```

und vergleichen Sie MATLABs Ergebnis `x=A\b` mit unserem Spielzeug-Programm. Meistens sollten die Ergebnisse gleich sein. Testen Sie viele Systeme und geben Sie an: in wieviel Prozent der Fälle versagt unser naives Programm? Warum?

Ü 4.2 Fehlerempfindlichkeit im Eliminationsverfahren

Aufgabe 33: Fehlerempfindlichkeit

Lösen Sie mit Matlabs Bergabstrich das Gleichungssystem $A\mathbf{x} = \mathbf{b}$ (es ist das Gleichungssystem Nr. 4 aus Aufgabe 31),

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Addieren Sie nun zur Matrix künstliche Fehler-Matrizen der Form $\delta A =$ `0.01*rand(4)` und vergleichen Sie die Lösung mit jener des ungestörten Systems.

Berechnen Sie die relativen Fehler der Matrixdaten[22] $||\delta A||/||A||$ und der Lösung $||\delta \mathbf{x}||/||\mathbf{x}||$. Testen Sie einige Fälle, auch mit kleineren oder größeren Störtermen, und schätzen Sie daraus das *maximale Verhältnis* von rel. Fehler der Lösungs zu rel. Fehler in den Daten ab.

Was bedeutet in diesem Zusammenhang der Wert `cond(A)`?

Ändern Sie nun die Matrix des Gleichungssystems zu

$$A = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix}$$

(Hinweis: `hilb(4)` liefert genau diese Matrix!) und wiederholen Sie Ihre Untersuchungen.

Ü 4.3 Entwicklung der Determinante, Komplexität

Aufgabe 34: Determinante

Gegeben sind

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix}, D = \begin{bmatrix} 10 & 9 & 8 & 7 \\ 6 & 5 & 4 & 3 \\ 2 & 1 & 0 & -1 \\ -2 & -3 & -4 & -5 \end{bmatrix}$$

Berechnen Sie von diesen Matrizen die Determinante auf verschiedene Arten:

1. Mit der Matlab-Funktion `det( )`
2. durch *LR*-Zerlegung, Matlab-Funktion `lu( )`, aus dem Produkt der Diagonalelemente von R[23]. Der Befehl `diag(R)` extrahiert übrigens die Hauptdiagonale einer Matrix als einen Vektor, der Befehl `prod` multipliziert alle Elemente.
3. Auch das primitive *LR*-Programm aus Aufgabe 32 liefert die Determinante als Produkt der Diagonalelemente (sogar vorzeichenrichtig, weil es keine Pivot-Suche durchführt). Testen Sie!
4. Die folgende Funktion implementiert das Standardverfahren zur Berechnung der Determinante durch Entwicklung nach Unterdeterminanten. (Copy-Paste aus PDF, lässt sich auch von der Übungs-Homepage herunterladen)

```
function d = mydet(A)
n = length(A);
if n==1
    d = A(1,1);
else
    d = 0;
```

[22] Sie brauchen Matrixnormen dazu! Informieren Sie sich in Skript und Vorlesungsfolien!

[23] Vorsicht, es kann sein, dass Matlabs `lu()` während der Rechnung Zeilen der Matrix vertauscht (Pivotierung). Mit jedem Zeilentausch wechselt das Vorzeichen der Determinante. Hat also `lu()` eine ungerade Anzahl von Zeilen vertauscht, hat das Produkt der Diagonalelemente das falsche Vorzeichen. Im Rahmen dieser Aufgabe sind Vorzeichen Glückssache.

```
        sig = 1;
        for i=1:n
            d = d + sig*A(1,i)*mydet(A(2:n,[1:i-1,i+1:n]));
            sig = -sig;
        end
    end
```

Wiederholen Sie auch die bekannten Standardverfahren (Regel von Sarrus, Entwicklung nach Unterdeterminanten) und rechnen Sie für A, B, C von Hand nach.

> Achtung: die Regel von Sarrus (Gitterzaunregel) gilt nur für 2×2- und 3×3-Matrizen. Wer 4×4-Determinanten auf diese Weise berechnen will, rechnet falsch. FALSCH! **FALSCH!!**

Aufgabe 35:

Ein Rechenverfahren, das zwar korrekt rechnet, aber ewig dafür braucht, hat nur theoretischen Wert. Die Berechnung der Determinante durch Entwicklung nach Unterdeterminanten ist ein Beispiel dafür.

Die Funktion `mydet` aus Aufgabe 34 implementiert das Standardverfahren zur rekursiven Berechnung der Determinante durch Entwicklung nach Unterdeterminanten.

Testen Sie die Rechenzeit`mydet` im Vergleich zum MATLAB-Standardbefeht `det`. Einfache Testmatrizen liefert z. B. die Funktion `magic(n)`.

MATLAB bietet mit den Befehlen `tic` und `toc` eine Art Stoppuhr an, mit der sie die Rechenzeit messen können. Sie könnten dazu Code der folgenden Art verwenden:

```
>> A = magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> tic; mydet(A); toc
Elapsed time is 0.000203 seconds.
```

Versuchen Sie nun auch etwas größere Matrizen als in Aufgabe 34. Bis zu welchem n lässt sich die Determinante

1. in weniger als zehn Sekunden,
2. in weniger als einer Minute
3. in weniger als zehn Minuten

berechnen? Das hängt natürlich von der Leistungsfähigkeit Ihres Rechners ab. Schätzen Sie aufgrund Ihrer Zeitmessungen und der Tabelle im Skriptum, Kapitel 3.7.1, wie lange die Berechnung für $n = 15$ und $n = 20$ dauern würde.

Ü 4.4 Modell-Funktionen an Daten anpassen

Gegeben sind Datenpunkte, gesucht sind Parameter für ein Modell, das diese Daten beschreibt. Im einfachsten Fall: eine „schöne Kurve“, die sich dem Verlauf der Datenpunkte gut anpasst.

Statistisch aussagekräftig ist das nur, wenn (deutlich) mehr Datenpunkte gegeben als Modellparameter gesucht sind. Das führt auf überbestimmte Gleichungssysteme.

Material und Links dazu siehe Einleitung. Noch einmal vorweg die wichtige Aussage:

> MATLABs Bergabstrich \ findet für überbestimmte Gleichungssystemen (die in der Regel nicht exakt lösbar sind) ein „am wenigsten falsches “ Ergebnis.

Etwas exakter gesagt: MATLABs Bergabstrich \ findet für überbestimmte Gleichungssystemen die bestmögliche Näherung im Sinn der kleinsten Fehlerquadrate.

MATLAB bietet in seiner *curve fitting toolbox* viele Hilfsmittel, um Kurven oder Flächen an gegebene Daten anzupassen. Bevor Sie sich darin vertiefen, sollten Sie die grundlegenden Ideen und Methoden verstehen. Darum geht es in dieser Übungseinheit.

Ü 4.4.1 Eine Variable: Beispiel aus der MATLAB-Hilfe

Die MATLAB-Hilfe (R2019–R2021) diskutiert Beispiele unter MATLAB › Data Import and Analysis › Descriptive Statistics › Programmatic Fitting › MATLAB Functions for Polynomial Models (geben Sie *“Programmatic Fitting”* im Suchfenster der Hilfe ein, dann finden Sie dieses Beispiel rasch.)

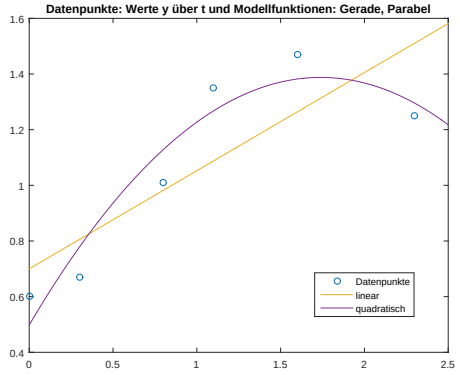
Hier ist das Beispiel im Abschnitt *MATLAB Functions for Polynomial Models* für Sie aufbereitet, so dass Sie aus den Angaben der MATLAB-Hilfe und den Zusatzinforrmationen folgende Aufgabe lösen können:[24]

Gegeben sind Messwerte y zu verschiedenen Zeiten t als Spaltenvektoren:

```
t = [0 0.3 0.8 1.1 1.6 2.3]';
y = [0.6 0.67 1.01 1.35 1.47 1.25]';
```

Sie sollen eine Funktion finden, die den Zusammenhang $y = y(t)$ gut modelliert. Einfache Ansätze sind: eine lineare Funktion, oder ein allgemeineres Polynom, zum Beispiel:

$$y = a_0 + a_1 t \qquad \text{oder}$$
$$y = a_0 + a_1 t + a_2 t^2$$



Dafür gibt es in MATLAB die Befehle `polyfit` und `polyval` und das *Basic Fitting User Interface*. Tipp: lassen Sie mit `plot` die Punkte zeichnen. Klicken Sie im *figure*-Fenster auf Tools › Basic Fitting und probieren Sie aus!

[24] Eigentlich müssen Sie nur die Beispiel-Befehle aus der Matlab-Hilfe in Ihre eigene Skript-Datei kopieren. Aber bitte schalten Sie ihr Hirn nicht aus, während Sie Strg + C und Strg + V verwenden!

Die Polynomfunktionen passen sich in diesem Beispiel nicht besonders gut an die Daten an. Versuchen Sie stattdessen ein lineares Modell[25] mit allgemeineren Funktionen, nämlich

$$y(t) = a_0 + a_1 e^{-t} + a_2 t e^{-t}$$

Vorgangsweise: Setzen Sie die gegebenen Datenpunkte in die Ansatzfunktion ein. Sie erhalten pro Wertepaar eine (lineare!) Gleichung in den drei Unbekannten a_0, a_1, a_2.

$$\begin{aligned} 0{,}60 &= a_0 + a_1 e^{0{,}0} + a_2 \cdot 0{,}0 \cdot e^{0{,}0} \\ 0{,}67 &= a_0 + a_1 e^{-0{,}3} + a_2 \cdot 0{,}3 \cdot e^{-0{,}3} \\ 1{,}01 &= a_0 + a_1 e^{-0{,}8} + a_2 \cdot 0{,}8 \cdot e^{-0{,}8} \\ \dots &\quad \dots \end{aligned}$$

Im MATLAB-Skript erstellen Sie die Koeffizientenmatrix X (die MATLAB-Hilfe sagt: *form the design matrix*). Achtung: `t` und `y` müssen Spaltenvektoren sein!

```
X = [ones(size(t))  exp(-t)  t.*exp(-t)];
```

Der Aufbau dieser Matrix ist der entscheidende Punkt, den Sie verstehen müssen. Die Koeffizienten von a_0 im obigen Gleichungssystem sind immer eins, daher enthält die erste Spalte von X lauter Einser, wie ein Volksschulzeugnis. Die Koeffizienten von a_1 sind e^{-t}-Werte, daher steht in der zweiten Spalte von X ein Vektor von e^{-t}-Werten, und so weiter.

Bitte mitdenken: Natürlich ist nicht immer automatisch die erste Spalte von X ein Volksschulzeugnis – das hängt von den gewählten Ansatzfunktionen ab, und von deren Reihenfolge. Aber als einfache Regel gilt: „Die Spalten von X enthalten die Werte der Ansatzfunktionen."

Die rechte Seite sind einfach nur die y-Werte. Die Modellkoeffizienten a_0, a_1, a_2 berechnet der Bergabstrich-Operator \ als bestmögliche Näherung aus dem überbestimmten Gleichungssystem:

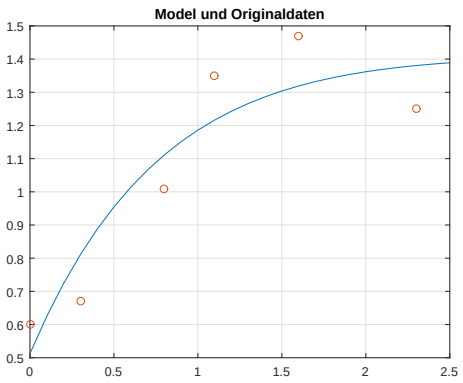
```
a = X\y
a =

    1.3983
   -0.8860
    0.3085
```

Also ist das bestmögliche Datenmodell gegeben durch

$$y = 1{,}3983 - 0{,}8860\, e^{-t} + 0{,}3085\, t e^{-t}.$$

Jetzt können Sie Ihre Modellfunktion auswerten (in genügend hoher Auflösung, also an vielen t-Punkten in engem Abstand) und zusammen mit den Originaldaten zeichnen.

```
T = (0:0.1:2.5)';
Y = [ones(size(T))  exp(-T)  T.*exp(-T)]*a;
plot(T,Y,'-',t,y,'o'), grid on
title('Modell und Originaldaten')
```



[25] *linear* sind hier nicht die verwendeten Ansatzfunktionen e^{-t} und te^{-t}. *Lineares Modell* bedeutet, dass diese Funktionen *linear kombiniert* werden: also ein Ansatz der Form „Koeffizient 1 mal Funktion 1 plus Koeffizient 2 mal Funktion 2…"

Auch hier bitte mitdenken: das Einsetzen von t-Daten in das Modell geschieht elegant wieder in Matrix-Vektor-Form: Die Matrix Y ist ähnlich aufgebaut wie die vorher verwendete X-Matrix. Multiplikation von Y mit Koeffizientenvektor $\mathbf{a}$ liefert Modellwerte.

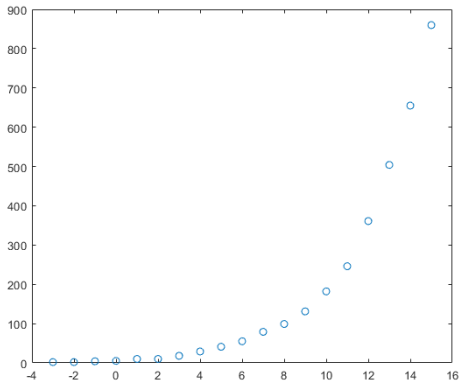
Alternative Auswertung: (in Wirklichkeit dieselbe Rechnung wie oben, nur für die a-Vektorkomponenten einzeln angeschrieben – vielleicht ist es so leichter verständlich)

```
Y = [ones(size(T))  exp(-T)  T.*exp(-T)]*a;  % Auswertung in Matrix-Vektor-Form
%
Y = a(1) + exp(-T)*a(2) + T.*exp(-T)*a(3); % Alternativ: komponentenweise Schreibung
```

Ü 4.4.2 Anpassen eines exponentiellen Wachstumsmodells

Gegeben sind (t, y)-Wertepaare. (MATLAB-Quelltext, copy-paste-fähig, nur die einfachen Anführungszeichen bei `'o'` müssen Sie ausbessern!)

```
t = -3:15;
y = [2 2 4 5 10 10 18 29 41 55 79 ...
99 131 182 246 361 504 655 860];
plot(t,y,'o')
```



Gesucht sind die Parameter a und b eines exponentiellen Wachstumsmodells

$$y = a \exp(bt) \ .$$

Es handelt sich bei y um die COVID-19-Fälle in Österreich (aufsummiert), mit Zeitachse t in Tagen vom 26. Februar bis 15. März 2020.

Wenn Sie hier, so wie vorher im Abschnitt Ü 4.4.1, die y- und t-Werte in die Modellgleichung einsetzen, erhalten Sie pro Wertepaar eine Gleichung für die unbekannten Parameter a und b.

$$\begin{aligned} 2 &= a \exp(-3b) \\ 2 &= a \exp(-2b) \\ 4 &= a \exp(-b) \\ \vdots &\qquad \vdots \\ 860 &= a \exp(15b) \end{aligned}$$

Im Unterschied zu vorher sind die Gleichungen jedoch nichtlinear in a und b. Wir behandeln echt nichtlineare überbestimmte Systeme erst später. Dieses System lässt sich aber zu einem linearen Problem vereinfachen.

Dieser einfache Trick zeigt dir, wie du ein nichtlineares System loswirst. . .

Logarithmieren Sie die Gleichungen:

$$\begin{aligned} \log 2 &= \log a - 3b \\ \log 2 &= \log a - 2b \\ \log 4 &= \log a - b \\ &\vdots \qquad\quad \vdots \\ \log 860 &= \log a - 15b \end{aligned}$$

(Natürlich steht log hier für den natürlichen Logarithmus!) Nennen Sie noch $\log a = \bar{a}$, dann steht hier ein überbestimmtes lineares Gleichungssystem in den Unbekannten $\bar{a}$ und b.

In MATLAB transformieren Sie die Datenvektoren von Zeilen- zu Spaltenvektoren. Die Koeffizientenmatrix X und rechte Seite dieses Systems erzeugen Sie dann so

```
t = t';
y = y';
X = [ones(size(t)) t];
rhs = log(y);
```

Der Bergabschrägstrich liefert die kleinste-Quadrate-Schätzung für die unbekannten Parameter $\bar{a} = \log a$ und b.

```
ab=X\rhs
```

Für die hier gegebenen Daten ist `ab(1)` $= \bar{a} = \log a = 1.7602$ und damit der Parameter $a = \exp(\bar{a}) = 5.8135$. Das ist der (vom Modell geschätzte) Anfangswert für $y(0)$, wobei hier $t = 0$ der 29. Februar ist.

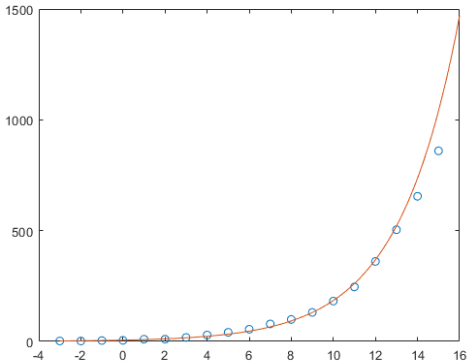
Der Wert `ab(2)` $= b = 0.3458$ ist die Wachstumskonstante im exponentiellen Modell. Um den Faktor $\exp(b) = 1.4132$ multiplizieren sich pro Tag die y-Werte, das entspricht einem täglichen Zuwachs um 41%. Die Verdopplungszeit T_2 in Tagen erhalten Sie aus

$$T_2 = \frac{\log 2}{b} = 2.0043 \quad .$$

Erinnern Sie sich noch, das war die beunruhigende Datenlage vor drei Jahren. Schauen wir uns die Daten und das geschätzte Wachstumsmodell genauer an.

Wir brauchen für die Zeitachse Zeitpunkte in feinerer Auflösung und gleich mal einen Tag in die Zukunft extrapoliert. `linspace` liefert hundert Zeitpunkte, das reicht. Für diesen `T`-Vektor werten wir das Modell aus.

```
a = exp(ab(1));
b = ab(2);
T = linspace(-3,16)';
Y = a*exp(b*T);
plot(t,y,'o',T,Y)
```



Zum Glück liegen die letzten beiden Datenpunkte schon unter der Modellkurve.

Vorsicht! Daraus allein hätte sich (mit Datenstand von März 2020) keine fundierte Vorhersage ableiten lassen. Dazu sind das Modell und unsere Datenanpassung zu stark vereinfacht. Was sich damals aussagen ließ: Ein exponentielles Wachstumsmodell mit den hier berechneten Parametern sieht in der grafischen Darstellung – bis auf die letzten beiden Datenpunkte – plausibel aus; **wenn sonst nichts das Wachstum bremst**, sagt dieses Modell in zwei Wochen katastrophale Fallzahlen voraus.

Im Nachhinein lassen sich die Modell-Voraussagen überprüfen. Am Ende dieses Abschnittes ist dazu eine grafische Darstellung bis Anfang April ergänzt.

Vorher versuchen wir aber noch eine etwas verfeinerte Modellrechnung. Denn unser logarithmierender Linearisierungs-Trick verursacht einen statistischen Fehler: die Parameter-Schätzung minimiert die Fehler in den logarithmierten Daten. Dadurch werden die (weniger aussagekräftigen) Datenpunkte zu Beginn stärker gewichtet als die (aktuelleren) Daten gegen Ende des Zeitintervalls.

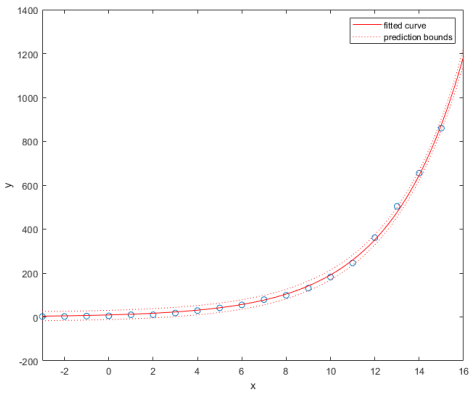
Wir werden uns noch eingehender mit der direkten Anpassung des nichtlinearen Datenmodells $y = a\exp(bt)$ befassen, aber jetzt lassen wir MATLAB arbeiten. Wenn bei Ihnen die *Curve Fitting Toolbox* installiert ist, funktioniert der Befehl
`modell = fit(t,y,'exp1')` Achtung, da steht 1, eins, nicht klein-ell bei `'exp1'` !)
und liefert als Ergebnis

```
modell =
     General model Exp1:
     modell(x) = a*exp(b*x)
     Coefficients (with 95% confidence bounds):
       a =       9.162  (7.889, 10.44)
       b =       0.304  (0.294, 0.314)
```

Zu den geschätzten Parametern wird auch ein 95%-Konfidenzintervall angegeben. **Vorsicht!** Diese Angaben gehen davon aus, dass die Daten tatsächlich einem exponentiellen Gesetz gehorchen und die einzelnen Datenpunkte nur zufallsbedingt mit einer gewissen Wahrscheinlichkeitsverteilung von der exponentiellen Kurve abweichen. Die Aussage ist also: *Wenn* ein exponentielles Wachstum vorläge, *dann wären das* die geschätzten Parameter.

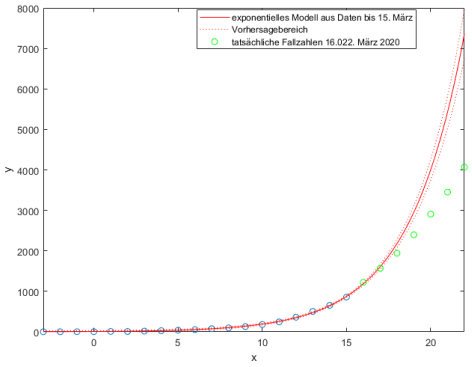
Wenn das Modell über MATLABs `fit`-Befehl berechnet ist, gibt es zusätzliche Optionen für die graphische Darstellung: Das Schlüsselwort `'predobs'` zeichnet auch Unsicherheits-Grenzen ein. Der `xlim([-3,16])`-Befehl stellt den x- (hier t-) Achsenbereich ein; ein größerer Endwert würde das Modell weiter in die Zukunft hin extrapolieren.

```
plot(t,y,'o')
hold on
xlim([-3,16])
plot(modell,'predobs')
hold off
```



Nochmal **Vorsicht!** Die engen Unsicherheitsgrenzen sollen nicht den Eindruck hoher Genauigkeit oder Verlässlichkeit entstehen lassen. Die Kurven fußen auf der Annahme, dass der Verlauf wirklich einem exponentiellen Modell folgt.

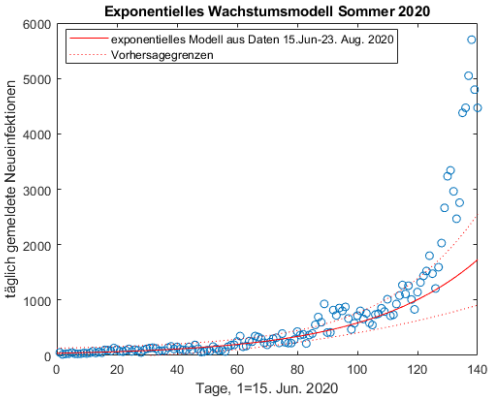
Wir haben das Modell in der obigen Graphik nicht weiter in die Zukunft extrapoliert, weil die Modellannahmen stark vereinfacht und unsicher waren. Tatsächlich konnte der erste Lockdown das exponentielle Wachstum bremsen. Die mit mehr Datenpunkten ergänzte Grafik rechts zeigt, dass einige Tage nach Beginn des Lockdowns die gemeldeten Fallzahlen von der exponentiellen Modellkurve abweichen. Es stellt sich ein anderes Wachstumsmodell ein.



Aufgabe 36: COVID-19 im Sommer und Herbst 2020

Sie können hier Daten zu den täglichen Neuinfektionen in Österreich, Zeitraum 140 Tage (15. Juni bis 1. Nov 2020) herunterladen.

Berechnen Sie aus den ersten 70 Datenpunkten die Parameter eines exponentiellen Wachstumsmodells, sowohl in der einfachen Form (linearer Fit an die logarithmierten Daten), als auch mit MATLABS *curve fitting toolbox*, Befehl `fit`. Vergleichen Sie das angepasste exponentielle Modell mit den tatsächlichen Daten. Stellen Sie Ihre Modellrechnung etwa so dar, wie in der Grafik rechts.



Interpretation: Bis etwa Tag 120 liegen die tatsächlichen Fallzahlen einigermaßen innerhalb der Vorhersagegrenzen, aber dann brechen die Daten nach oben aus. Offensichtlich hatte sich zu diesem Zeitpunkt (Mitte Oktober) der Ausbreitungsmodus signifikant verändert. Konsequenter Weise mussten ab Anfang November wieder Ausgangsbeschränkungen verordnet werden.

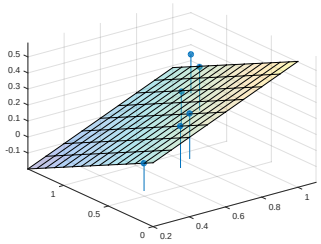
Ü 4.4.3 Mehrere Variable

Die MATLAB-Hilfe zeigt *Multiple Regression* als nächstes Beispiel unter
MATLAB » Data Import and Analysis » Descriptive Statistics » Programmatic Fitting » Multiple Regression .

Beschreibung und Illustration siehe auch 4. Vorlesung, vorletzte Folie. Auch das Skriptum erklärt in Kapitel 5.4: Anpassen eines linearen Modells (einer Ausgleichsebene).

Hier eine Musterlösung zur Aufgabe der vorletzten Vorlesungsfolie. Im Vergleich zur Lösung in der MATLAB-Hilfe zeigt sie auch das Zeichnen der Ebene und der Originaldaten.

```
%% Datenpunkte: Variable x1, x2 und Messwert y
x1 = [.2 .5 .6 .8 1.0 1.1]';
x2 = [.1 .3 .4 .9 1.1 1.4]';
y  = [.17 .26 .28 .23 .27 .24]';

%% Matrix des ueberbestimmten Systems
% fuer  Ansatz y = a0 + a1*x1 + a2*x2
X = [ones(size(x1))  x1  x2];

%% Loesung: Koeffizientenvektor
a = X\y
%% Auswertung und Zeichnung auf feinem Gitter
[XX,YY]=meshgrid(0.2:0.1:1,0:0.1:1.4);
ZZ= a(1) + a(2)*XX + a(3)*YY;

surf(XX,YY,ZZ,'FaceAlpha',0.3),
axis tight, hold on
% Originaldaten
stem3(x1,x2,y,'.','MarkerSize',25)
hold off
```



Ü 4.5 Weitere Aufgaben zu Linearen Datenmodellen

Aufgabe 37: Hochwasser

Für die Blies (einen Nebenfluss der Saar) sollen die Hochwasserstände am Pegel Neunkirchen aus den Wasserständen des Pegels Ottweiler und des Pegels Hangard vorhergesagt werden. Es liegen die Daten der Scheitelwasserstände von 12 Winterhochwässern aus den Jahren 1963–1971 vor: (Daten können Sie von der Übungs-Homepage runterladen)

Wasserstand in cm												
Neunkirchen y	172	309	302	283	443	298	319	419	361	267	337	230
Ottweiler x_1	93	193	187	174	291	184	205	260	212	169	216	144
Hangard x_2	120	258	255	238	317	246	265	304	292	242	272	191

Quelle: U. Maniak, Hydrologie und Wasserwirtschaft, Springer, 1988

Wir unterstellen den Daten das lineare Modell $a_0 + a_1 x_1 + a_2 x_2 = y$. In diesem Ansatz sind a_0, a_1 und a_2 unbekannte Koeffizienten, die aus den zwölf gegebenen Werte-Tripeln möglichst gut bestimmt werden sollen.

Berechnen Sie die Koeffizienten des linearen Modells, und geben Sie auch den Differenzenvektor zwischen Modellvorhersage und Messwerten an. (Dieses Beispiel ist im Skriptum Kapitel 5.4 ausführlich durchgerechnet)

Aufgabe 38: Jack Sparrows Kompass

Egal, ob Sie eine Geophysik-Vorlesung besucht oder Johnny Depp in *Pirates of the Caribbean* gesehen haben, Sie haben gelernt, dass eine Kompassnadel nicht immer nach Norden zeigt.

Die Abweichung heißt magnetische Deklination, ihre Kenntnis ist nach wie vor von großer Wichtigkeit für die Seefahrt. Deswegen gibt die Zentralanstalt für Meteorologie und Geodynamik regelmäßig für alle Landeshauptstädte aktuelle Werte der magnetischen Deklination an:[a] (Daten von Übungs-Hompage herunterladbar!)

[a] Update 2017: Diese Tabelle wird von der ZAMG inzwischen nicht mehr bereitgestellt, vermutlich, weil die Bedeutung der Seefahrt in den Landeshauptstädten abgenommen hat)

Deklinationswerte der einzelnen Landeshauptstädte
(östliche Deklination bezogen auf Jahresmitte 2008)

Stadt	Länge x	Breite y	Deklination D
Wien (WIK)	16.37	48.20	3.0000
Eisenstadt	16.52	47.85	3.0333
St.Pölten	15.63	48.20	2.9000
Graz	15.45	47.07	2.7833
Linz	14.30	48.30	2.5500
Klagenfurt	14.31	46.62	2.5000
Salzburg	13.03	47.80	2.2500
Innsbruck	11.40	47.27	1.8833
Bregenz	9.77	47.50	1.4000

Finden Sie für diese Daten eine Anpassung der Form

$$z(x, y) = a_1 + a_2 x + a_3 x^2 + a_4 y$$

Geben Sie die Differenz $D - z(x, y)$ zwischen den tatsächlichen Deklinationswerten und den angepassten Werten an. Welcher Wert wird am schlechtesten approximiert?

Moderne Modelle für das Erdmagnetfeld (International Geomagnetic Reference Field IGRF, World Magnetic Model WMM) verwenden im Prinzip denselben Ansatz: Sie passen ein Modell an Messdaten an. Aber im Unterschied zu unserem Polynom-Ansatz für 9 Datenpunkte verarbeiten Sie riesige Datenmengen und verwenden als Ansatz Kugelflächenfunktionen.

Aufgabe 39: Für Weicheier

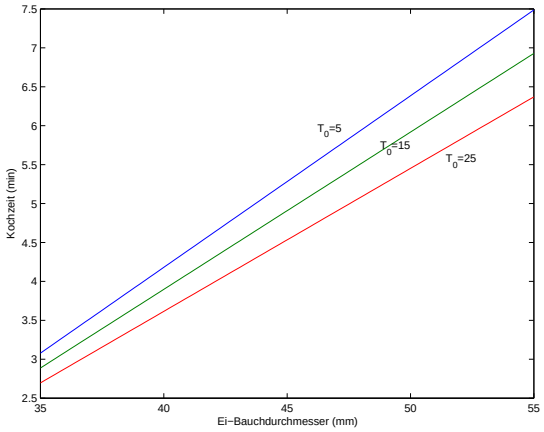
Der Wiener Physiker Werner Gruber beschäftigt sich mit der Thermodynamik des Eierkochens und gibt nebenstehende Werte für die Kochzeit eines perfekt weichen Frühstücks-Eis an, abhängig von Bauchdurchmesser d (nicht seiner, der des Eis) und Ausgangstemperatur T_0 (Daten können Sie von der Übungs-Homepage runterladen)

Durchmesser d (mm)	Ausgangstemperatur T_0 (C)	Kochzeit (min:sek)
35.0	4	3:10
40.0	4	4:10
45.0	4	5:20
50.0	4	6:30
35.0	20	3:00
40.0	20	3:40
45.0	20	4:40
50.0	20	5:50

Finden Sie für diese Daten eine Anpassung der Form

$$t(d, T_0) = a_1 + a_2 d + a_3 T_0 + a_4 d T_0$$

Zeichnen Sie ein Diagramm ähnlich dem hier gezeigten (x-Achse: Eidurchmesser, y-Achse: Kochzeit), in dem Sie für Anfangs/-temperaturen 5,15,25 Grad die Kochzeiten eintragen.

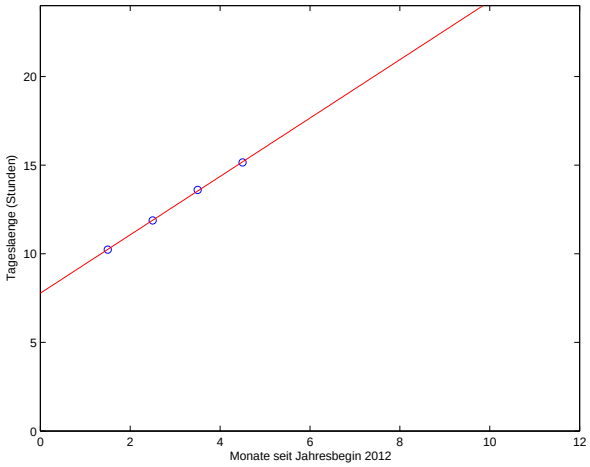


Aufgabe 40: Ein Reich, in dem die Sonne nicht untergeht

Eine Aufgabe zu Datenmodellen mit polynomialen und allgemeineren Ansatzfunktionen.

Die folgende Tabelle gibt für Leoben die Tageslänge (von Sonnenauf- bis -untergang) in Stunden für vier Tage von Februar bis Mai an. Klar ersichtlich ist ein eindeutiger Trend zu immer längeren Tagen.

Datum kalendarisch	Monate seit Jahresbeg.	Tageslänge Stunden
15. Feb. 2019	1,5	10,23
15. Mär. 2019	2,5	11,88
15. Apr. 2019	3,5	13,60
15. Mai. 2019	4,5	15,15



Anlageberater und andere begabte Verkäufer könnten Ihnen anhand der Grafik zu diesen Daten überzeugend erklären, dass spätestens Ende Oktober die Sonne nicht mehr untergeht.

Bevor Sie nun in Solar-Aktien investieren, erstellen Sie selber Modelle für diesen Datensatz

1. Lineare Regression: $y = a_0 + a_1 x$
2. Polynom-Ansatz $y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$
3. Datenmodell $y = a_1 + a_2 \cos\left(x\frac{\pi}{6}\right) + a_3 \sin\left(x\frac{\pi}{6}\right)$

Stellen Sie die Datenpunkte und die drei Modelle in einer Grafik dar (Achsen-Bereich wie oben). Beantworten Sie (durch Ablesen aus der grafischen Darstellung) folgende Fragen: Ab wann geht laut Modell 1 die Sonne in Leoben nicht mehr unter (Tageslänge 24 h)? Für wann sagt Modell 2 ewige Nacht voraus (Tageslänge 0 h)? Wann ist laut Modell 3 Sommersonnenwende (Tageslänge maximal)?

Nur Modell 3 kann verlässliche Voraussagen treffen, weil es einen problemgerechten Ansatz verwendet: die periodischen Schwankungen des Sonnenlaufes werden durch Sinus- und Cosinusterme genähert.