

# Matlab-Übungen zu Deterministische Signale und Systeme Musterlösung zur 2. Übung

## 1. Aufgabe

In Matlab ist es möglich auf einzelne Elemente einer Matrix  $A$  zuzugreifen. Dies kann man dadurch erreichen, indem man den Ausdruck `A(Zeile, Spalte)` verwendet. Man übergibt dabei formal Vektoren, die die zu selektierenden Zeilen bzw. Spalten als Elemente enthalten. Ist z. B. die Matrix

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}$$

gegeben, so wählt `A([1],[1])` oder auch die Schreibweise `A(1,1)` das Element links oben, also  $a_{11}$ , aus. `A([1 3],[2 3 4])` gibt die Matrix

$$\begin{pmatrix} a_{12} & a_{13} & a_{14} \\ a_{32} & a_{33} & a_{34} \end{pmatrix}$$

zurück.

Es gibt weitere Angaben, die in Klammern hinter einer Variablen stehen dürfen und eine Sonderbedeutung haben. Das sind im Einzelnen

**"end"** welches die letzte Zeile bzw. Spalte selektiert und

**":"** der alle Zeilen bzw. alle Spalten selektiert. Die Angabe `A(:, :)` gibt z. B. die ganze Matrix  $A$  aus.

Vektoren können dabei auch in der Form `1:10` bzw. `2:2:8` übergeben werden. `A(1:2:5, 3:end)` gibt z. B. die Matrix

$$\begin{pmatrix} a_{13} & a_{14} & a_{15} \\ a_{33} & a_{34} & a_{35} \\ a_{53} & a_{54} & a_{55} \end{pmatrix}$$

aus.

Der **":"** Operator kann in der Notation `(:)` verwendet werden um alle Elemente eines Arrays als Komponente eines einzelnen Spaltenvektors zu betrachten. Dabei werden nacheinander die

einzelnen Indizes »durchgezählt«, d.h. bei einer Matrix werden die Spalten aneinander gehängt.  $A(:)$  gibt bspw. den Vektor

$$(a_{11} \ a_{21} \ a_{31} \ a_{41} \ a_{51} \ a_{12} \ a_{22} \ a_{32} \ a_{42} \ a_{52} \ a_{13} \ a_{23} \ \cdots \ a_{45} \ a_{55})^T$$

aus.

Das Maximum über alle Einträge der Matrix  $A$  erhält man z. B. mit  $\max(A(:))$ .

Die Verwendung von  $(:)$  oder  $(:).'$  kann über dies dazu verwendet werden um einen Vektor unabhängig von seiner ursprünglichen Ausrichtung als Spalten- oder Zeilenvektor anzusprechen.

Die genannten Schreibweisen können nicht nur zum Selektieren von Elementen genutzt werden, Elemente oder Bereiche von Vektoren und Matrizen können auf diese Weise auch gesetzt werden. Die Zuweisung  $A(3,3) = 7$  setzt das Element  $A_{33}$  auf den Wert 7.

- 1.1. Ein »Magisches Quadrat« ist eine quadratische Matrix ( $N \times N$ ), deren Zeilensummen, Spaltensummen und Diagonalsummen gleich sind. Dabei dürfen die Zahlen von 1 bis  $N^2$  aber nur jeweils genau einmal vorkommen. In Matlab kann man ein Magisches Quadrat der Größe  $N \times N$  mit  $\text{magic}(N)$  erzeugen. Speichern Sie ein Magisches Quadrat der Größe  $8 \times 8$  in der Variablen  $A$ .
- 1.2. Erzeugen Sie einen Zeilenvektor, der hintereinander die Zeilenvektoren der Matrix  $M$  enthält.
- 1.3. Setzen Sie alle Elemente von  $A$  mit einem nur aus geraden oder nur aus ungeraden Zahlen bestehenden Index auf den Wert 0.
- 1.4. Ersetzen Sie die obere linke »Ecke« von  $A$  durch ein Magisches Quadrat der Größe  $4 \times 4$ . Verfahren Sie mit den verbleibenden drei »Ecken« der Matrix genauso. Ist die  $4 \times 4$ -Matrix, die sich aus den mittleren 16 Elementen von  $A$  zusammensetzt, auch wieder ein Magisches Quadrat? Sie können zum Überprüfen die Funktion  $\text{sum}(A)$ , die Ihnen die Spaltensummen von  $A$  ausgibt und die Funktion  $\text{diag}(A)$ , die Ihnen die Diagonalelemente von  $A$  ausgibt, verwenden.

## Lösung

- 1.1. Die Zeile

```
A = magic(8); %Erzeugung 8x8-Matrix
```

speichert das Magische Quadrat in der Variablen  $A$ . Man kann leicht überprüfen, ob es sich wirklich um ein magisches Quadrat handelt, indem man die Spalten-, Zeilen und Diagonalsummen berechnet und das Vorkommen der Zahlen 1 bis  $N \cdot M$  überprüft, wobei  $M$  und  $N$  die Anzahl der Zeilen bzw. Spalten der Matrix meint:

```
sum(A)
sum(A.')
sum(diag(A))
sum(diag(A')) %Sollte alles das gleiche Ergebnis liefern
```

```

for kk = 1 : numel(A) %Anzahl der Elemente in A
    if isempty(find(A == kk)) %Suche den Zahlenwert kk in A
        error('Kein magisches Quadrat!') %Wenn leer, Zahlenwert nicht gefunden
    end
end
end

```

Das Ergebnis der Summenbildung ist jeweils 260.

#### 1.2. Mit

```
B = [ A(1, :) A(2, :) A(3, :) A(4, :) ...
```

erzeugt man den gewünschten Zeilenvektor. Die Berechnung mit

```
tic;
B = A';
```

ist deutlich kompakter und unabhängig von der Dimension der Matrix.

#### 1.3. Mit

setzt man alle Kombinationen ungerader Indexziffern auf 0 und mit

```
A(1 : 2 : 7, 1 : 2 : 7) = 0;
```

erfolgt das gleiche für gerade Ziffern. Zur Kontrolle gibt man die Matrix **A** aus:

```
A(2 : 2 : 8, 2 : 2 : 8) = 0;
```

#### 1.4. Analog zur vorangegangenen Aufgabe ersetzt man die vier Bereiche der Matrix durch ein Magisches Quadrat:

```
A(1 : 4, 1 : 4) = magic(4);
```

```
A(5 : 8, 1 : 4) = magic(4);
```

```
A(1 : 4, 5 : 8) = magic(4);
```

In Matrix **B** wird der gewünschte Ausschnitt gespeichert:

Schließlich wird wieder mit den Spaltensummen

den Zeilensummen

```
sum(B)
```

den beiden Diagonalsummen

```
sum(B.')
```

```
sum(diag(B))
```

(das Ergebnis der Summenbildung ist jeweils 34) und dem »Nachzählen« der vorkommenden Zahlen

```

for k = 1 : prod(size(B))
    if isempty(find(B == k))
        error('Kein mag. Quadrat!')
    end
end

```

überprüft, ob die Bedingungen erfüllt sind.

## 2. Aufgabe

Funktionen stehen in Matlab in der Regel in einer eigenen Datei. Dabei muss die Datei den gleichen Namen wie die Funktion tragen.

Ein typisches Grundgerüst einer Funktion zeigt der folgende Code:

```

function [a,b] = tollefunktion (c,d)
% TOLLEFUNKTION Meine Funktion macht etwas ganz tolles ! Dabei
% bekommt sie die folgenden Optionen uebergeben :
%
% c: erster Parameter
% d: zweiter Parameter
%
% Die folgenden Werte werden zurueckgegeben :
%
% a: erster Rueckgabewert
% b: zweiter Rueckgabewert
%

a = c / d; %Quellcode der Funktion!
b = c * d;

end

```

Eine Besonderheit von Matlab ist die Fähigkeit, daß Funktionen mehrere Werte auf einmal zurückgeben können.

Dem Kommentar direkt unter der ersten Zeile, in der die Funktion deklariert wird, kommt eine Sonderbedeutung zu. Gibt man auf der Kommandozeile **help funktion** ein (und die Datei, die die Funktion enthält, befindet sich im Suchpfad), wird dieser Kommentar ausgegeben.

- 2.1. Schreiben Sie eine Funktion **betrag**, die den Betrag einer Zahl (eines reellen Skalars) zurückgibt. Testen Sie hierfür, ob die übergebene Zahl größer/gleich oder kleiner Null ist und bilden Sie einen entsprechenden Rückgabewert. Geben Sie der Funktion auch eine Beschreibung, die mit **help betrag** ausgegeben werden kann und überprüfen Sie die Funktion und die Hilfe zur Funktion.
- 2.2. Schreiben Sie eine Funktion **parallel**, die zwei Impedanzen als Eingabewerte übergeben bekommt und als Ergebnis die Impedanz der Parallelschaltung dieser Impedanzen zurückgibt.

- 2.3. Erweitern Sie die Funktion aus 2.2 so, dass ein Vektor von mehreren Impedanzwerten übergeben werden kann und deren Parallelschaltung berechnet wird.
- 2.4. Schreiben Sie eine entsprechende Funktion `seriell`, die gleiches für die Reihenschaltung leistet.
- 2.5. Verwenden Sie verschachtelte Funktionsaufrufe, um die Impedanzen der Schaltungen aus Abbildung 1 zu berechnen (bei einer Frequenz von 2,5 GHz).

*Hinweis: Es kann der Übersichtlichkeit dienen, die Zehnerpotenzen der Einheiten als »Pseudoeinheiten« anzulegen und zu verwenden. Definiert man z.B.  $\text{kHz} = 1\text{e}3$ ; und  $\text{nH} = 1\text{e}-9$ ;, lässt sich das Berechnen einer Impedanz als  $Z = j*2*\pi * 3*\text{kHz} * 50*\text{nH}$ ; aufschreiben.*

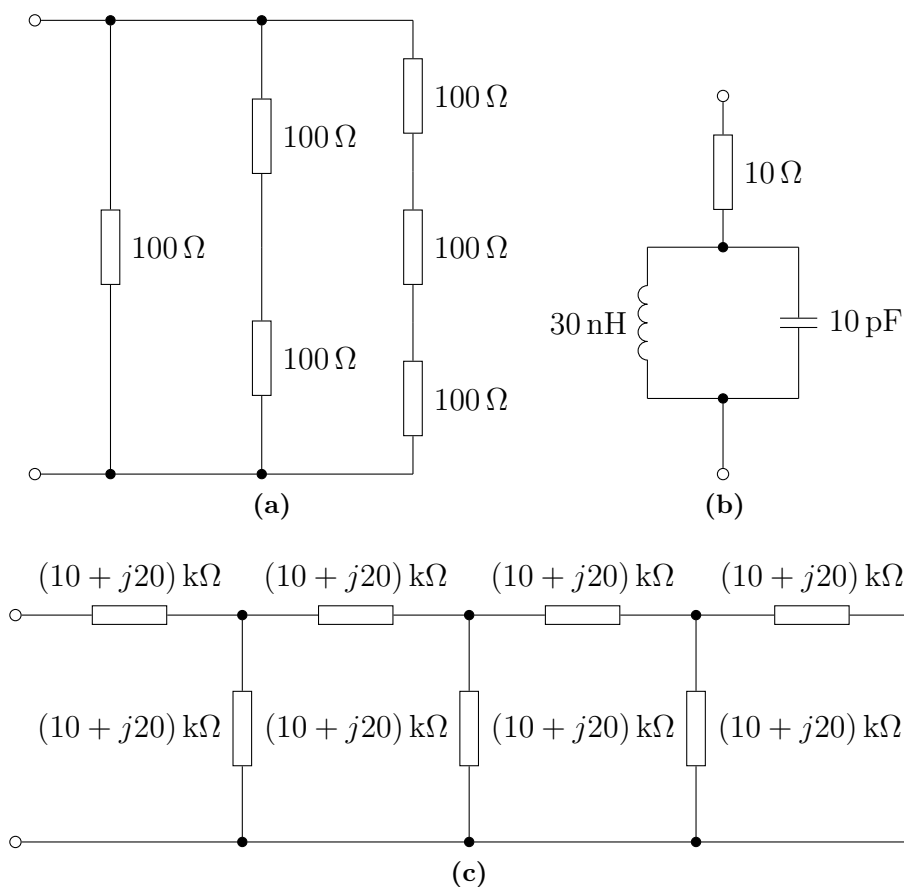


Abbildung 1: Schaltungen.

## Lösung

2.1. Die Datei `betrag.m`:

```
function out = betrag(in)
% BETRAG berechnet den Betrag des uebergebenen Wertes
```

```

if in >= 0
    out = in;
else
    out = -in;
end

```

- 2.2. Die Datei `parallel_two.m` (hier wurde ein anderer Name gewählt, um die Datei von der Lösung der nächsten Aufgabe unterscheiden zu können):

```

function zout = parallel_two(zina, zinb)
% PARALLEL berechnet die Parallelschaltung der Impedanzwerte der
%   Eingabewerte.
%
%   zina, zinb: Impedanzwerte, deren Parallelschaltung berechnet
%               werden soll
%   zout:       Das Ergebnis der Parallelschaltung

zout = 1 / (1 / zina + 1 / zinb);

end

```

- 2.3. Um das Problem effizient zu lösen, macht man sich zunutze, daß die Parallelschaltung einer Summenbildung der Admittanzen entspricht. Durch diesen Trick kann man die `sum`-Funktion von Matlab nutzen, wodurch man langsame Schleifen vermeidet. Die Datei `parallel.m`:

```

function zout = parallel(zin)
% PARALLEL berechnet die Parallelschaltung der Impedanzwerte des
%   Eingabevektors.
%
%   zin: Vektor, der komplexe Impedanzwerte enthaelt
%   zout: Skalar, der das Ergebnis der Parallelschaltung enthaelt

yin = 1 ./ zin;
yout = sum(yin);
zout = 1 / yout;

```

- 2.4. Die Funktion `seriell` muß einfach nur die Eingangswerte aufsummieren:

```

function zout = seriell(zin)
% SERIELL berechnet die Reihenschaltung der Impedanzwerte des
%   Eingabevektors.
%
%   zin: Vektor, der komplexe Impedanzwerte enthaelt
%   zout: Skalar, der das Ergebnis der Reihenschaltung enthaelt

zout = sum(zin);

```



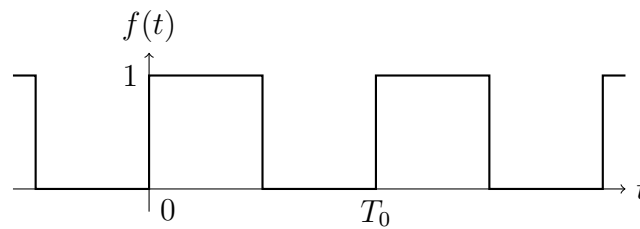


Abbildung 2: Rechtecksignal.

### 3. Aufgabe

Die Fourierreihe des Rechtecksignals aus Abbildung 2 ist gegeben durch

$$f(t) = \frac{1}{2} + \frac{2}{\pi} \sum_{n=0}^{\infty} \frac{1}{2n+1} \sin((2n+1)2\pi f_0 t). \quad (1)$$

Da der Rechner naturgemäß seine Probleme mit dem Zählen bis  $\infty$  hat, ersetzen wir Gleichung 1 durch

$$f(t) = \frac{1}{2} + \frac{2}{\pi} \sum_{n=0}^N \frac{1}{2n+1} \sin((2n+1)2\pi f_0 t), \quad (2)$$

welche für  $N \rightarrow \infty$  in zuvor genannte Gleichung übergeht.

- 3.1. Schreiben Sie ein Matlab-Script, das ein Rechtecksignal nach Abbildung 2 mit der Periode  $T_0 = 2$  erzeugt. Die Zeitachse soll auf  $0 \leq t \leq 4$  begrenzt sein. Wählen Sie eine Zeitauflösung von 0,005. Plotten Sie dieses Signal und stellen Sie dabei (mit Hilfe der Befehle `xlim` und `ylim`) einen auf jeder Seite um 0,2 vergrößerten Bereich dar.
- 3.2. Schreiben Sie eine Matlab-Funktion mit dem Namen `fourierreihe`, die einen Zeitvektor `t` und einen Wert für  $f_0$  und  $N$  übergeben bekommt und die Annäherung gemäß der in Gleichung 2 gegebenen Fourierreihe berechnet und zurückgibt.
- 3.3. Erweitern Sie Ihr in 3.1 geschriebenes Skript: Verwenden Sie die Funktion `fourierreihe`, um die Annäherung an  $f(t)$  darzustellen. Verwenden Sie eine `for`-Schleife und den Befehl `pause`, um die Originalfunktion und die Annäherung in einem gemeinsamen Plot nacheinander für alle  $N$  von 0 bis 20 darzustellen (`pause` unterbricht das laufende Programm und wartet darauf, dass Sie die Eingabetaste drücken). Zeigen Sie  $N$  dabei im Titel des Plots an.

### Lösung

- 3.1. Sicher gibt es mehrere Möglichkeiten, ein Rechtecksignal zu erzeugen. Eine Möglichkeit:

```
% Zeitachse erstellen
t = 0 : .005 : 4;
T0 = 2;
```



```
% Rechtecksignal erstellen
f = zeros(1,length(t));
for k = 1 : length(t)
    if mod(floor(t(k)), T0) == 0 %Modulo bezogen auf T0
        f(k) = 1;
    end
end
```

Alternativ zu der `for`-Schleife kann man das Rechtecksignal auch folgendermaßen generieren:

```
f = ~mod(floor(t), T0);
```

Hier sei noch einmal darauf hingewiesen, dass Schleifen in Matlab oft vermieden werden können und dass diese Lösung fast immer vorzuziehen sind, weil sie kompakter und vor allem deutlich schneller sind. Vor allem beim Programmieren von Algorithmen, die selbst iterativ arbeiten, ist es wichtig den vielfach asugeführten Kern des Algorithmus möglichst effizient zu implementieren.

Die Darstellung des Rechtecksignals kann mit

```
figure;
plot(t, f);
xlim([-0.2 4.2]);
ylim([-0.2 1.2]);
```

erzeugt werden.

Hinweis: Der Befehl `figure` öffnet ein neues Plotfenster. Will man dieses später noch einmal benutzen, z. B. um einen weiteren Graphen in das gleiche Fenster zu zeichnen, kann man dem Plotfenster auch eine numerische Kennung geben, etwa `figure(3)`. Ruft man dann (später) ein weiteres Mal `figure(3)` auf, wird dieses Fenster wieder »aktiv« und man kann an diesem Plot weiterarbeiten. Man kann Plotfenstern aber auch einen Variablennamen zuweisen (z. B. `myfigure = figure;`), um später wieder darauf zugreifen zu können.

### 3.2. Die Datei `fourierreihe.m`:

```
function out = fourierreihe(t, f0, N)

% Gleichanteil a0/2 festlegen
out = 0.5 * ones(1,length(t));

% Grund- und Oberschwingungen aufsummieren
for n = 0 : N
    out = out + 2/pi/(2*n+1)*sin((2*n+1)*2*pi*f0*t);
end
```

Da die Funktion `sin(x)` für einen Vektor `x` den Sinus für jedes Element von `x` zurückgibt, braucht man keine Schleife, um über die Zeitachse zu iterieren.

### 3.3. Der erweiterte Teil des Skripts `Aufgabe3.m`:

```
figure;  
for k = 0 : 20  
    g = fourierreihe(t,.5,k);  
    plot(t,f,t,g);  
    xlim([-0.2 4.2]);  
    ylim([-0.2 1.2]);  
    title(['N=' num2str(k)]);  
    pause %Pause wartet bis zum naechsten Tastendruck  
end
```

Um den Zahlenwert für das aktuelle  $N$ , der in  $k$  gespeichert ist, im Titel anzeigen zu können, wird er zuerst mit `num2str` zu einem String konvertiert.

## 4. Aufgabe

Sie möchten überprüfen, ob Ihre Gitarre richtig gestimmt ist. Dazu haben Sie eine Aufnahme des Tons »a« erzeugt und möchten nun die Frequenz herauslesen und zur Kontrolle die Aufnahme mit einem am Rechner generierten Sinuston der Frequenz 440 Hz vergleichen. (In einer späteren Übung werden wir eine »schönere« als die hier vorgestellte Methode kennenlernen.) Die Abtastfrequenz Ihrer Aufnahme beträgt 96 kHz.

- 4.1. Lesen Sie Ihre Gitarrenaufnahme mit dem Import-Tool von Matlab in die Variable `gitarre` ein (die Datei mit der Aufnahme finden Sie in der beigelegten zip-Datei). Stellen Sie diese Aufnahme dann als »plot« dar. Achten Sie dabei darauf, dass die  $x$ -Achse die richtigen Werte anzeigt. Erzeugen Sie hierzu aus den Ihnen bekannten Größen einen geeigneten Zeitvektor.

Hinweis: Das Import-Tool finden Sie im Menü unter »File→Import Data«. Dort können Sie dann die Datei auswählen und genauere Einstellungen zum Datei-Import machen.

Die Datei `gitarre.csv` ist im CSV-Format gespeichert, das z. B. von vielen Tabellenkalkulationsprogrammen verwendet werden kann. In Octave können Sie zum Einlesen der Daten die Funktion `dlmread('dateiname')` verwenden (und auch Matlab bietet diese Funktion).



- 4.2. Schätzen Sie nun die Frequenz Ihres Gitarren-Signals ab, indem Sie die Zeit zwischen zwei Nulldurchgängen des Signals zählen. Ist Ihre Gitarre richtig gestimmt?

Hinweis: Beim Abzählen der Frequenz könnte Ihnen der Befehl `grid` helfen.

- 4.3. Legen Sie nun zur Kontrolle eine Sinusschwingung der Frequenz 440 Hz über den Plot und vergleichen Sie die beiden Schwingungen (die Amplitude der Sinus-Schwingung sollte in der gleichen Größenordnung wie die des Gitarrensignals liegen). Legen Sie dazu erst eine Variable mit dem Vergleichssignal an. Es sollte die gleiche Anzahl an Meßwerten wie das Gitarrensignal aufweisen (und natürlich auch die gleiche Abtastfrequenz). Plotten Sie nun das zweite Signal in den ersten Plot, indem Sie den Matlab-Befehl `hold on` verwenden. Warum erzeugt Ihre Gitarre keine reine Sinusschwingung wie Sie sie mit Matlab erzeugt haben?

4.4. Sie möchten nun den Plot etwas verschönern, um ihn auszudrucken und Ihrer Band zu zeigen. Setzen Sie zuerst die Plot-Funktion von Matlab mit **hold off** wieder zurück.

Plotten Sie nun die beiden Kurven in *einem einzigen plot*-Befehl. Sorgen Sie in diesem Befehlsaufruf auch gleich dafür, daß das Gitarrensignal in rot und das Referenzsignal in blau und gestrichelt dargestellt wird.

Geben Sie dem Plot nun eine sinnvolle Achsenbeschriftung für beide Achsen (für die  $x$ -Achse etwa »t in s«). Geben Sie dem Diagramm auch einen Titel und fügen Sie ein Gitter und eine Legende hinzu, die deutlich macht, welches das Gitarren- und welches das Referenzsignal ist.

Speichern Sie den entstandenen Plot nun einmal im Matlab-Grafikformat (Dateiendung: **.fig**) und einmal als PDF-Datei zum Drucken ab.

Plotten Sie außerdem einen Ausschnitt des Signals, der nur die positive  $y$ -Achse und etwas mehr als eine Signalperiode auf der  $x$ -Achse zeigt, um die Frequenz der Gitarre genauer abzählen zu können.

## Lösung

4.1. Zunächst ist es sinnvoll, grundlegende Angaben aus der Aufgabenstellung in Variablen zu speichern:

```
fs = 96e3; % Samplingfrequenz , 96kHz
ts = 1 / fs; % Laenge eines Samples
```

Danach kann die Datei eingelesen werden:

```
gitarre = dlmread('gitarre.csv'); %Einlesen der Daten
```

Um zu plotten, muß noch die Zeitachse erstellt werden, mit deren Hilfe das Signal dann zeitrichtig dargestellt wird.


```
% Laenge der Aufnahme in Sekunden
% Achtung, Matlab beginnt bei 1 zu zaehlen!
T = (length(gitarre)-1) * ts;
```

```
t = 0 : ts : T; % Zeitachse
```

```
figure;
plot(t, gitarre);
```

4.2. Das Gitter erzeugt man mit

```
grid;
```

Zum Ablesen können auch sog. »Data Cursor« hilfreich sein, die man über die Symbolleiste () oder das Menü erreicht.

Aus den beiden Nulldurchgängen bei  $t_1 = 2,32 \text{ ns}$  und  $t_2 = 4,65 \text{ ns}$  ergibt sich eine Frequenz von

$$f = \frac{1}{T} = \frac{1}{t_2 - t_1} \approx 429 \text{ Hz.}$$

Die Gitarre ist also nicht richtig gestimmt.

#### 4.3. Die Sinusschwingung, die man gemäß

```
sinus = 1.3 * sin(2*pi*440*t);  
hold on;  
plot(t, sinus);
```

erzeugt, zeigt, daß die Gitarre nicht richtig gestimmt ist. Der Befehl **hold on** sorgt dabei dafür, daß der Plotbefehl den schon vorhandenen Graphen nicht überschreibt sondern den zweiten Graphen zusätzlich ins gleiche Fenster zeichnet. Mit **hold off** schaltet man dieses Verhalten wieder aus.

#### 4.4. Der Code zum »Aufhübschen« der Diagramme sieht wie folgt aus:

```
hold off;  
  
figure;  
plot(t, gitarre, 'r', t, sinus, 'b--');  
  
xlabel('t/s');  
ylabel('Amplitude');  
title('Gitarre');  
grid;  
legend('Gitarre', 'Referenzsignal');  
  
saveas(gcf, 'gitarre.fig'); % Speichern als Matlab-Figure  
saveas(gcf, 'gitarre.pdf'); % Speichern als PDF  
  
xlim([0 3e-3]); % Ausschnittsvergroesserung  
ylim([0 2]);  
  
saveas(gcf, 'gitarre-ausschnitt.fig'); % Speichern als Matlab-Figure  
saveas(gcf, 'gitarre-ausschnitt.pdf'); % Speichern als PDF
```

Dabei wird dem Plot-Befehl über den String 'r' mitgeteilt, daß der erste Graph rot gezeichnet werden soll und 'b--' sorgt dafür, daß der zweite Graph blau und gestrichelt erscheint. Einen Überblick über alle Möglichkeiten der »Liniengestaltung« gibt die Hilfe mittels **doc LineSpec** aus. In Octave bekommt man mittels **help plot** eine kleine Übersicht über diese Möglichkeiten.

Alternativ kann das Bild auch im Menü über File→Save as abgespeichert werden.

