v1.2

# Matlab-Übungen zu Deterministische Signale und Systeme Musterlösung zur 4. Übung

## 1. Aufgabe

Ab dieser Aufgabe wird nicht mehr explizit darauf hingewiesen, wann es während des Programmierens der Lösungen sinnvoll ist, eine Funktion zu plotten, um die korrekte Funktion der Programme zu überprüfen. Entscheiden Sie selbst, wann das für Sie sinnvoll ist. Plots, die Ergebnisse dokumentieren, werden aber weiterhin in der Aufgabenstellung erwähnt.

In der folgenden Aufgabe soll die Faltung zweier Rechteckfunktionen berechnet werden.

1.1. Zum Lösen dieser Aufgabe benötigen Sie zunächst eine Zeitachse mit $t \in [-2\,\mathrm{s}, 3\,\mathrm{s}]$ bei einer Abtastfrequenz von 100 Hz. Legen Sie die Zeitachse an. Erzeugen Sie nun einen Rechteckimpuls innerhalb dieses Zeitbereichs mit

$$r_1(t) = \begin{cases} 1 & 0\,\mathrm{s} \leq t \leq 1\,\mathrm{s} \\ 0 & \text{sonst} \end{cases}$$

und einen Zweiten Rechteckimpuls mit

$$r_2(t) = \begin{cases} 1 & 0\,\mathrm{s} \leq t \leq 2\,\mathrm{s} \\ 0 & \text{sonst.} \end{cases}$$

1.2. Schreiben Sie eine Funktion `cval`, die zwei Signale und deren Zeitachsen übergeben bekommt und die Faltung der beiden Signale berechnet. Die Integration soll dabei durch eine einfache Summenbildung der Funktionswerte realisiert werden, so dass aus dem Faltungsintegral

$$f(t) = \int_{-\infty}^{\infty} f_1(\tau) f_2(t-\tau) d\tau$$

der zeitdiskrete Ausdruck

$$f(n) = T_s \sum_{k=-\infty}^{\infty} f_1(k) f_2(n-k).$$

wird. $T_s$ ist hierbei der Abstand zwischen zwei Zeitwerten und entspricht $d\tau$ im Faltungsintegral.

M. Sc. M. Trinh Hoang (S108/207); K. Cui
Prof. Dr.-Ing. A. Klein
Prof. Dr.-Ing. M. Pesavento

Wintersemester 2018/19

Berechnen Sie nur Werte, die Sie aus den beiden Funktionen auch tatsächlich bestimmen können. (Überlegen Sie sich dafür zunächst, wie eine passende neue Zeitachse aussehen könnte, und implementieren Sie dann die Funktion zur Faltung.)

*Hinweis: Funktionswerte, die nicht gegeben sind, können, falls nötig, als 0 angenommen werden.*

1.3. Schreiben Sie nun eine weitere Funktion `ctime`, der Sie zwei Funktionen und deren Zeitachsen übergeben und die eine neue Zeitachse für das Ergebnis der Faltung der beiden Funktionen erzeugt.

Überlegen Sie hierzu, wie man den Anfangszeitpunkt und die Länge des neuen Zeitachsenvektors bestimmen kann.

1.4. Benutzen Sie die beiden Funktionen `ctime` und `cval` um die Faltung der beiden Signale aus Aufgabenteil 1.1 zu berechnen und grafisch darzustellen. Berechnen Sie das Ergebnis zusätzlich »von Hand« und überprüfen Sie damit das Ergebnis aus Matlab.

1.5. Lösen Sie entsprechend Aufgabenteil 1.4 auch die Aufgaben 1 und 4 der 5. Übung zu »Deterministische Signale und Systeme«. Verwenden Sie einen Abstand von 0,1 zwischen zwei Samples. Stimmt die Lösung mit der erwarteten überein oder ergibt sich ein Fehler? Falls sich ein Fehler ergibt, bestimmen Sie dessen Ursache und beschreiben Sie, wie er sich vermeiden lässt.

# Lösung

1.1. Mit dem Speichern der gebenenen Werte

```
fs = 100; %Sampling-Frequenz
Ts = 1/fs;
```

und dem Anlegen der Zeitachse

```
t = -2 : Ts : 3;
```

legt man in zwei Schritten

```
r1 = zeros(1, length(t)); %Mit Nullen gefuellter Vektor
r2 = r1;

r1(t>=0 & t<=1) = 1; %Rechteck erzeugen (logischer Index!)
r2(t>=0 & t<=2) = 1;
```

die beiden Funktionen an.

1.2. Um die Faltung entsprechend der Aufgabenstellung (Integral durch Summenbildung annähern) zu implementieren ist außer den beiden Funktionen eigentlich nur eine Zeitachse vonnöten, da sie nur benutzt wird, um die Samplingfrequenz zu bestimmen. Diese könnte man natürlich auch direkt übergeben. Trotzdem werden »der Vollständigkeit halber« beide Zeitachsen übergeben, auch in Hinblick auf eine spätere Aufgabe:

```
function f = cval(t1, f1, t2, f2)
```

Nun bestimmt man den Abtastabstand

```
Ts = t1(2) - t1(1);
```

Die Funktionen liegen aber beide nur in einem bestimmten Zeitintervall vor, wodurch die Grenzen der Summe eingeschränkt werden. Da aber nicht für jeden Schritt Werte von beiden Funktionen vorhanden sind, wie Abbildung 1 zeigt, müssen die Funktionen entsprechend mit Nullen aufgefüllt werden, um die vollständige Summe zu bilden.

Eine etwas elegantere Methode erhält man, wenn man nur die eine Funktion an den Rändern mit Nullen auffüllt und jeweils nur einen der Länge der anderen Funktion entsprechenden Ausschnitt für die Summenbildung wählt. Damit hat man die nicht gegebenen Werte der anderen Funktion auch explizit auf Null gesetzt. Noch eleganter wird es (zumindest in Matlab), wenn man die Summe durch das Skalarprodukt zweier Vektoren ersetzt. Formal kann man die Faltung von zwei abgetasteten Funktionen $f_1$ und $f_2$, dargestellt durch Vektoren mit den Längen $n$ bzw. $m$, als Matrix-Vektor Produkt darstellen. Das Ergebnis ist eine Funktion als Vektor

$$\begin{pmatrix} f(1) \\ f(2) \\ \vdots \\ f(l-1) \\ f(l) \end{pmatrix} = \begin{pmatrix} f_1(1) & 0 & \dots & 0 & 0 \\ f_1(2) & f_1(1) & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & f(n) & f_1(n-1) \\ 0 & 0 & \dots & 0 & f_1(n) \end{pmatrix} \cdot \begin{pmatrix} f_2(1) \\ f_2(2) \\ \vdots \\ f_2(m-1) \\ f_2(m) \end{pmatrix}$$

mit $l = n + m - 1$ Elementen. Die Zeilen der Matrix sind Auschnitte der invertierten Funktion $f_1$, die skalar mit dem Vektor $f_2$ multipliziert das jeweilige Element des Vektors $f$ ergeben. Die vollstängige Matrix in Matlab aufzubauen macht wenig Sinn, da es viel Speicher kostet, die Komplexität jedoch nicht verringert. Deswegen werden die Zeilen der Matrix in Form eines Hilfsvektors `x` aufgebaut und jedes Element einzeln berechnet. Die vollständige Faltung wird wie beschrieben realisiert:

```
x = zeros(1, length(f2));

% Laenge des Ergebis' der Faltung
l = length(f1) + length(f2) - 1;

% Allozieren des Ergebinsvektors
f = zeros(1, l);

for k = 1 : l
    % Aubauen des Arbeitsvektors
    if k <= length(f1) % Wenn Laufindex kleiner als f1...
        x = [f1(k) x(1 : end-1)]; %...Elemente von f1 nachschieben ...
    else
        x = [0 x(1 : end-1)]; %... ansonsten Nullen nachschieben
    end
```

```
    % Bilden der Faltunssumme zu einem speziellen Zeitpunkt
    f(k) = x * f2' * Ts;
end
```

1.3. Die Funktion `ctime` soll vier Parameter übergeben bekommen und einen Wert zurückliefern, was zur Funktionsdefinition

```
function t = ctime(t1, f1, t2, f2)
```

führt. Der zurückgegebe Parameter `t` wird die berechnete Zeitachse in Form eines Vektors enthalten. Als Berechnungsgrundlage dienen dafür die Zeitachse `t1` und Funktionswerte `f1` der ersten Funktion und entsprechend auch `t2` und `f2`, Zeitachse und Werte der zweiten Funktion.

Wenn man davon ausgeht (was wir hier tun), dass beide Zeitachsen die gleiche Abtastfrequenz benutzen, reichen prinzipiell auch beide Zeitachsen oder eine Zeitachse und beide Funktionen, um die neue Zeitachse zu erstellen, aber so ist man auf jeden Fall auf der sicheren Seite.

Das weitere Vorgehen ist die Berechnung des Abtastabstandes

```
Ts = t1(2) - t1(1);
```

das Berechnen des Anfangszeitpunktes

```
Tl = t1(1) + t2(1);
```

und der Gesamtlänge der neuen Zeitachse in Samples

```
l = length(t1) + length(t2) - 1;
```

und zuletzt dem Bilden der neuen Zeitachse

```
t = Tl : Ts : Tl+(l-1)*Ts;
```

Die Gesamtlänge der neuen Zeitachse ergibt sich dabei aus der Aneinanderreihung der beiden gegebenen Zeitachsen, wobei eine um ein Sample verkürzte Achse entsteht, da nur dann Werte »generiert« werden, wenn mindestens ein Wert der beiden Funktionen Anteil am aktuellen Schritt haben.

Abbildung 1 stellt den ersten und den letzten Schritt (sowie einen Zwischenschritt) der Faltungsoperation dar, um diese Beziehung zu veranschaulichen (jeweils die eine Funktion oben und die gespiegelte, verschobene Funktion unten). Hier wurde eine Auflösung von 1 Sample pro Sekunde gewählt.

An der Abbildung kann man auch direkt Start- und Endwert der neuen Zeitachse ablesen, der erste Wert ergibt sich für $t_1 = -2$ und $t_2 = -1$, was zu einem Wert für $t = -3$ führt. Der letzte Wert wird folglich bei $t = 5$ erzeugt.

Die Idee, erst linke und rechte Grenze der Zeitachse zu berechnen und dann dazwischen mit Werten im Abtastabstand aufzufüllen ist zwar grundsätzlich möglich, scheitert aber daran, dass Matlab aufgrund von Rundungsfehlern die zweite Grenze nicht erreicht und damit einen Punkt zuwenig erzeugt:

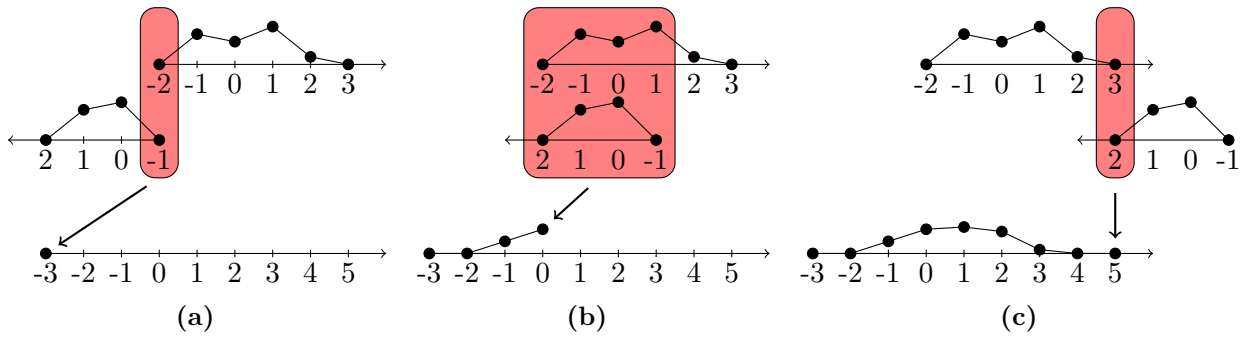


**Abbildung 1:** Veranschaulichung der Faltung mittels abgetasteter Funktionen.

```
%Tr = t1(end) + t2(end);
%format long;
%t = Tl : Ts : Tr;
```

Trotzdem kann die Methode verwendet werden. Matlab stellt hierfür den Befehl `linspace` zur Verfügung, dem man die beiden Grenzen und die Anzahl der Punkte übergibt und der einen entsprechenden Vektor zurückgibt.

1.4. Da die entsprechenden Funktionen schon im ersten Aufgabenteil angelegt wurden, stellt man mittels

```
plot(ctime(t, r1, t, r2), cval(t, r1, t, r2));
```

die Faltung der beiden Funktionen grafisch dar.

1.5. Die beiden Signale zu Aufgabe 1 sind ein Rechteck und ein Dreieck, die man mit

```
tx = 0 : .1 : 2.9;
th = 0 : .1 : 1.9;

x = ones(1, length(tx));
h = .5*th;
```

gemeinsam mit ihren Zeitachsen anlegt.

```
figure;
plot(ctime(tx, x, th, h), cval(tx, x, th, h));
grid;
```

stellt das Ergebnis der Faltung grafisch dar.

Bei Aufgabe 5 verfährt man entsprechend:

```
t = 0 : .1 : .9;
x = ones(1, length(t));
h = -x;

figure;
```

```
plot(ctime(t, x, t, h), cval(t, x, t, h));
grid;
```

Hier ergibt sich nun das Problem, dass die Werte beider Ergebnisse nicht mit der Theorie übereinstimmen. Dafür gibt es zwei Gründe. Beim Übergang vom Faltungsintegral zur Summe wird bei einem Überlapp der Funktionen $f_1$ und $f_2$ von $n \cdot T_s$ über $n+1$ Stützstellen summiert (siehe Abbildung 1). Hier wird also »zu viel« aufsummiert. Zusätzlich wird das Integral durch die Ober- bzw. Untersummen approximiert.

Da beide Fehler linear in der Abtastzeit $T_s$ sind, läßt sich die Genauigkeit durch die Wahl kleinerer Zeitschritte kontrollieren.

# 2. Aufgabe

Sie haben in dieser und der vergangenen Übung nun Funktionen geschrieben, mit deren Hilfe Sie die Fouriertransformation und die Faltung durchführen können.

Gegeben seien das Signal und das System aus Abbildung 2. Die Abtastfrequenz beträgt 100 Hz.

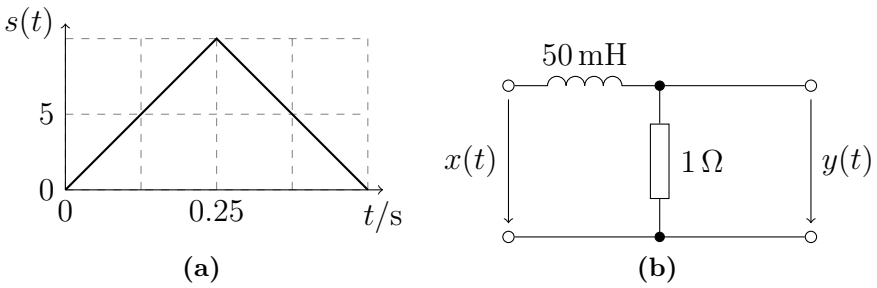


**Abbildung 2:** Signal und System.

2.1. Berechnen Sie die Impulsantwort des Systems aus der Abbildung von Hand. Um was für ein System handelt es sich? Wie wirkt sich das System auf Eingangssignale aus?

2.2. Das Signal $s(t)$ wird nun als Eingangssignal für das System aus Abbildung 2 verwendet. Berechnen Sie das Ausgangssignal $y(t)$ einmal mittels Fouriertransformation und ein zweites Mal mittels Faltung.

Verwenden Sie hierfür eine Zeitachse mit $t \in [-1\,\mathrm{s}, 1\,\mathrm{s}]$, innerhalb dieser 400 Abtastwerte dargestellt werden sollen. Verwenden Sie weiter einen Frequenzbereich mit $\omega \in [-500\,\mathrm{s}^{-1}, 500\,\mathrm{s}^{-1}]$.

Für die Faltung können Sie die Funktion aus der vorangegangenen Aufgabe verwenden. Benutzen Sie außerdem Ihre analytisch bestimmte Impulsantwort.

Für die Lösung mittels Transformation müssen Sie zuerst eine Funktion für die Fourierrücktransformation schreiben. Orientieren Sie sich dabei an der Lösung zur ersten Aufgabe der dritten Übung. Verwenden Sie zur Transformation des Signals in den Frequenzbereich die Funktion zur Fouriertransformation aus der dritten Übung.

Stellen Sie beide Lösungen gemeinsam mit den beiden Impulsantworten in einem Diagramm dar.

*Hinweis: Durch Rundungsfehler können Sie komplexe Zeitsignale bekommen, was Matlab beim Plotten mit einer Warnung quittiert. Das können Sie ignorieren, solange der komplexe Anteil klein gegenüber dem reellen ist.*

2.3. Berechnen Sie die »Fehlerleistung«, also die quadrierte Abweichung der beiden Lösungen voneinander in Abhängigkeit von der Zeit und in Abhängigkeit von der Frequenz und stellen Sie beide grafisch dar (Sie müssen dafür die erste Lösung in den Frequenzbereich transformieren, was natürlich für neue Fehler sorgt, dies soll hier aber nicht weiter stören). Die beiden Graphen sollen dabei untereinander in der gleichen Abbildung stehen. Verwenden Sie hierfür den Befehl `subplot`.

2.4. Bestimmen Sie die Zeit, die Matlab benötigt, um die Fouriertransformierte der Zeitbereichslösung in der vorangegangenen Aufgabe zu bestimmen. Bestimmen Sie die Zeit ein zweites Mal, diesmal aber mit 800 Abtastwerten. Wie hat sich die benötigte Zeit geändert?

# Lösung

2.1. Die Übertragungsfunktion der Schaltung ist

$$H(j\omega) = \frac{Y(j\omega)}{X(j\omega)} = \frac{R}{R + j\omega L} = \frac{R}{L}\frac{1}{\frac{R}{L} + j\omega}.$$

Aus der Tabelle in den Hilfsblättern kann man direkt die Lösung im Zeitbereich ablesen:

$$h(t) = \frac{R}{L} e^{-\frac{R}{L}t} u(t).$$

Das System ist ein Tiefpass. Eingangssignale werden vom System »geglättet«.

2.2. Der erste Schritt ist das Speichern der Bauteilewerte und der Anzahl der Stützstellen in Variablen

```
R = 1; % Ohm
L = 50e-3; % Henry
N = 400; % Anzahl Abtastpunkte
```

und das Anlegen von Zeit- und Frequenzachse

```
w = linspace(-500, 500, N);
t = linspace(-1, 1, N);
```

Für eine eventuelle spätere Verwendung wird noch der Abstand zwischen zwei Abtastwerten bestimmt:

```
Ts = t(2) - t(1);
```

Die Übertragungsfunktion

```
H = R./(R+j*w*L);
```

und die Impulsantwort des Systems

```
ho = R/L*exp(-R/L*t);
ho(t<0) = 0;
```

können nun direkt eingegeben werden. Die Einschaltfunktion $u(t)$ wurde dabei durch das Auf-Null-Setzen von Werten bei $t < 0$ gelöst. Sie können dafür aber auch mit der auf die Zeitachse angewendeten Funktion `heaviside` multiplizieren, die die Heaviside'sche Sprungfunktion implementiert.

**Lösung im Zeitbereich**

Das Signal bekommt seine eigene Zeitachse, um bei der Faltung nicht unnötig viele Nullwerte berechnen zu müssen:

```
ts = 0 : Ts : 0.5;
```

Nun stellt man sich das Signal aus zwei Geraden zusammen:

```
s = 40*ts;
s(ts > 0.25) = 10 - 40*(ts(ts > 0.25)-.25);
```

Auch hier kann man alternativ in Matlab eingebaute Funktionen verwenden: `triang(N)` erzeugt ein Dreieck der Länge $N$, wobei nur Werte $> 0$ erzeugt werden und `bartlett(N)` erzeugt ebenfalls ein Dreieck, nur diesmal mit den Nullwerten am Rand.

Die Faltung ist mit der Funktion aus der vorangegangenen Aufgabe leicht zu lösen:

```
y = cval(t, ho, ts, s);
ty = ctime(t, ho, ts, s);
```

Schließlich fehlt nur noch die grafische Darstellung

```
figure;
plot(t, ho, ts, s, ty, y);
grid;
xlabel('t/s')
```

des Ergebnisses.

**Lösung im Frequenzbereich**

Damit das in den Frequenzbereich transformierte Zeitsignal auf dem gleichen Frequenzbereich wie das System abgebildet wird, wird nun für beide die gleiche Zeitachse verwendet. Hierfür wird das Signal auf die »neue« Zeitachse angepasst:

```
s2 = zeros(1, length(t));
t0 = find(t>-Ts & t<Ts);
t0 = t0(1);
s2(t0 : t0+length(s)-1) = s;
```

Nach Transformation

```
S2 = ftrans(t', s2, w);
```

und Multiplikation

```
Y2 = S2 .* H;
```

bildet man die Lösung durch Rücktransformation in den Zeitbereich

```
y2 = frtrans(t', Y2, w);
```

Die dafür benötigte Funktion `frtrans` ist in der Datei `frtrans.m` zu finden.

Nun nur noch in das gleiche Diagramm plotten:

```
figure(1);
hold on;
plot(t, y2, '--');
hold off;
```

Wie schon in der Aufgabenstellung erwähnt, kann durch Rundungsfehler ein komplexes Zeitsignal entstehen, was allerdingfs keine physikalische Entsprechung hat. Solange es wirklich nur an Rundungsfehlern liegt, kann man den Imaginärteil des Signals also einfach nicht beachten. Der Plot-Befehl von Matlab bildet den Realteil, erzeugt aber eine Warnung. Möchte man diese auch verhindern, muß man selbst den Realteil bilden.

2.3. Um beide Graphen in einem Diagramm untereinander darzustellen, wird der Befehl `subplot` angewendet. Ihm werden die Gesamtzahl der Graphen, und die Anordnung des aktuellen Graphen in der Reihenfolge Spalte, Zeile übergeben. Den Fehler über der Zeit stellt man also mit

```
figure;
subplot(2,1,1)
plot(t, abs(y2-y(1 : length(y2))).^2);
grid;
xlabel('t/s');
```

dar und den Fehler im Frequenzbereich kann man mit

```
subplot(2,1,2)
plot(t, abs(Y2-ftrans(t', y(1 : length(y2)), w)).^2);
grid;
xlabel('w/s^{-1}');
```

darunter setzen.

2.4. Mittels `tic` und `toc` gibt Matlab die Zeit zum Berechnen der zwischen `tic` und `toc` liegenden Befehle aus. Auf meinem Rechner ergibt sich für

```
tic;
y = ftrans(t', y(1:length(y2)), w);
toc;
```

mit 400 Abtastwerten eine Dauer $t_1 \approx 0{,}22\,\mathrm{s}$. Bei 800 Abtastwerten benötigt mein Rechner $t_2 \approx 1{,}3\,\mathrm{s}$. Bei einer Verdoppelung der Abtastpunkte steit der Rechenaufwand um den Faktor 6. Berechnet man die Fouriertransformation auf diese Weise, ergibt sich schnell eine sehr große Dauer (und auch ein sehr großer Speicherbedarf). Es existieren jedoch Methoden, um die die Fouriertransformation wesentlich schneller und effizienter berechnen zu können, was in einer späteren Übung behandelt wird. Dies hat auch zur Folge, dass es im Allgemeinen deztlich effizienter ist im Frequnzbereich zu arbeiten.

# 3. Aufgabe

Matlab ist eine unter wenigen Programmiersprachen, bei der Funktionen mehrere Rückgabewerte haben können (ein weiteres prominentes Beispiel ist die Programmiersprache »Python«).

Um mehrere Werte zurückzugeben, wird bei der Funktionsdefinition statt einer Variablen einfach ein Vektor mehrerer Variablen als Rückgabewert angegeben.

```
function [r1, r2] = berechne(p1, p2, p3)
```

definiert also eine Funktion mit den beiden Rückgabewerten `r1` und `r2` und den drei übergebenen Parametern `p1`, `p2` und `p3`. In der Funktion müssen allen verwendeten Rückgabewerten Werte zugewiesen werden.

Beim Aufrufen der Funktion muss die Zuweisung auch an einen Vektor der entsprechenden Größe weitergegeben werden. Gemäß obigem Beispiel muss die Funktion als

```
[a, b] = berechne(1, 2, 3);
```

aufgerufen werden. Dabei wird der Rückgabewert `r1` in `a` und `r2` wird in `b` gespeichert.

3.1. Schreiben Sie eine Funktion `drahtwiderstand`, der Sie einen Skalar, einen String und einen zweiten Skalar übergeben. Die Funktion soll den Durchmesser, den Umfang, die Querschnittsfläche und den Widerstand eines runden Kupferdrahtes der Länge $l$ berechnen und alle vier Werte (in dieser Reihenfolge) zurückgeben. Dabei soll die Länge in Metern und ein beliebiger der drei restlichen geometrischen Werte (in Metern bzw. Quadratmetern) an die Funktion übergeben werden. Um zu unterscheiden, was übergeben wurde, soll der String verwendet werden. Wird als zweites Argument ein »d« übergeben, ist das nächste Argument der Durchmesser, wird ein »u« übergeben, ist der Umfang gemeint und ein »A« beschreibt die Fläche. Auch die Eingabe des Radius (»r« als Kennzeichen) soll möglich sein, allerdings soll intern nur mit dem Durchmesser gerechnet werden.

Ein Beispiel: Wird die Funktion als `drahtwiderstand(5, 'u', 3e-3)` aufgerufen, hat der Draht eine Länge von 5 m und einen Umfang von 3 mm und die restlichen Werte sollen berechnet werden.

*Hinweis: Eine mögliche Lösungsstrategie ist es, den Variablen* `u`*,* `d` *und* `A` *zunächst den Wert* `NaN` *zuzuweisen (was es mit* `NaN` *auf sich hat, sagt Ihnen die Hilfe) und dann die Variable für den gegebenen Wert zu überschreiben. Später kann man mit einer Abfrage testen, ob* `NaN` *oder ein Zahlenwert vorliegt (etwa mit der Funktion* `isnan`*) und so bestimmen, welche Werte berechnet werden müssen.*

Die spezifische Leitfähigkeit von Kupfer beträgt $58 \cdot 10^6\,\mathrm{S\,m^{-1}}$.

3.2. Welcher Wert wird zurückgegeben, wenn die Zuweisung an einen Skalar erfolgt (also `x = drahtwiderstand(...)`)? Ordnen Sie die zurückgegebenen Werte gegebenenfalls so an, daß in diesem Fall der Widerstand zurückgegeben wird.

3.3. Sie möchten dem Weihnachtsbaum im Garten eine Lichterkette spendieren. Um die Leitungsverluste zu berechnen, möchten Sie den Widerstand der Zuleitung wissen. Sie verwenden ein Kabel des Typs »H05VVF 3G0,75«, bei dem die Aderquerschnittsfläche 0,75 mm$^2$ beträgt. Der Baum befindet sich 50 m von der nächsten Steckdose entfernt.

# Lösung

3.1. Die Funktion `drahtwiderstand` beginnt mit der Funktionsdefinition

```
function [R, d, u ,A] = drahtwiderstand(l, type, val)
```

Die Besonderheit ist hier, dass vier Werte zurückgegeben werden.

Um später zu überprüfen, welcher Wert neben der Länge noch übergeben wurde, bedient man sich hier eines kleinen Tricks. Zuerst werden alle drei möglichen Werte mit `NaN` initialisiert. Das steht in Matlab für »Not a Number«. Es kann aber von vielen Matlab-Funktionen wie eine Zahl verarbeitet werden, und viele Funktionen liefern auch `NaN` statt einer Zahl zurück, z. B. wenn Sie einen einzelnen Wert in einem Vektor nicht, den Rest aber schon berechnen konnten. Hier nutzen wir `NaN` aber nur als Platzhalter, man hätte genausogut die Zahl 0 nehmen können, da sie ja offensichtlich als Eingabe auch zu keinem Sinnvollen ergebnis führt.

```
d = NaN;
u = NaN;
A = NaN;
```

Danach wird noch die Leitfähigkeit von Kupfer in einer Variablen gespeichert:

```
s_cu = 58e6;
```

Nun wird überprüft, welcher Wert gegeben ist, und dieser der entsprechenden Variablen zugewiesen. Falls eine ungültige Eingabe vorliegt, wird eine Fehlermeldung ausgegeben.

```
if strcmp(type, 'd')
    d = val;
elseif strcmp(type, 'r')
    d = 2*val;
elseif strcmp(type, 'u')
    u = val;
elseif strcmp(type, 'A')
    A = val;
else
    error('Ungueltige␣Eingabe!');
end
```

Nun kann man anhand der Überprüfung, ob eine Variable nicht gleich `NaN` ist, den anderen Variablen korrekte Werte zuweisen:

```
if ~isnan(d)
    A = pi/4*d^2;
    u = pi*d;
elseif ~isnan(u)
    d = u/pi;
    A = 1/4*u^2/pi;
else
    u = sqrt(4*pi*A);
    d = sqrt(4/pi*A);
end
```

Nun sind auch alle Werte vorhanden, und der Widerstand wird berechnet:

```
R = l/(A*s_cu);
```

Als kleine Besonderheit können wir nun noch überprüfen, ob als Eingabe für Radius, Durchmesser, Umfang oder Querschnitt eine 0 übergeben wurde. Falls dem so ist, ist der Widerstand des Drahtes nämlich unendlich hoch und da Matlab mit `Inf` den Wert unendlich kennt (und ihn viele Funktionen, ähnlich wie bei `NaN`, verarbeiten können), können wir diesen im Zweifelsfall auch zurückgeben:

```
if val == 0
    R = inf;
end
```

Alternativ zu `if elseif else` kann eine mehrseitige Bedingung in Matlab auch mit `switch case otherwise` implementiert werden. Außerdem könnte man auch direkt überprüfen, welchen String die Eingangsvariable `type` enthält. Die Funktion `drahtwiderstand` sieht dann folgendermaßen aus:

```
function [R, d, u ,A] = drahtwiderstand(l, type, val)
% DRAHTWIDERSTAND computes the impedance of a piece of copper wire.

s_cu = 58e6;

switch type
    case 'd'
        d = val;
        u = pi*d;
        A = pi*(d/2)^2;
    case 'u'
        u = val;
        d = u/pi;
        A = pi*(d/2)^2;
    case 'A'
        A = val;
        d = sqrt(A/pi)*2;
```

```
        u = pi*d;
    case 'r'
        d = 2*val;
        u = pi*d;
        A = pi*(d/2)^2;
    otherwise
        error('Ungueltige Eingabe!');
end

R = l/(A*s_cu);
end
```

3.2. Wird die Funktion ohne Zuweisung oder mit Zuweisung an einen Skalar aufgerufen, so wird stets der erste Rückgabewert zurückgegeben, in unserem Fall muß also `R` als erster Rückgabewert auftauchen.

Man kann so nicht nur einen oder alle Rückgabewerte »anfordern«, es werden stets so viele Rückgabewerte geliefert, wie im Ergebnisvektor Plätze dafür vorgesehen sind. Ein `[a, b] = drahtwiderstand(10, 'r', 1e-3)` liefert beispielsweise Widerstand und Durchmesser eine 10-m-Kabels mit dem Radius 1 mm.

3.3. Um den Widerstand des Kabels zu berechnen, übergeben wir unserer Funktion die Länge und den Querschnitt des Kabels, da wir beide Werte direkt gegeben haben. Hierbei hilft es zu beachten, daß unser 50-m-Kabel aus einem Hin- und einem Rückleiter besteht und so insgesamt 100 m zum Widerstand beitragen.

Mit

```
drahtwiderstand(2*50, 'A', .75e-6)
```

errechnen wir uns einen Widerstand von 2,3 Ω.

An dieser Stelle sei angemerkt, dass bei Funktionen, die mehrere Rückgabewerte besitzen auch einzelne verworfen werden können, z.B. m Speicher zu sparen. Dazu dient das `~` Symbol.

```
[~, d] = drahtwiderstand(1, 'A', pi*25)
```

verwirft den Widerstand und liefert nur den Durchmesser (10) zum Flächeninhalt ($\pi \cdot 5^2$).

# 4. Aufgabe

4.1. Kombinieren Sie die Funktionen zur Zeitachsenberechnung und zur Faltung aus der 1. Aufgabe zu einer neuen Funktion `tconv`, die Zeitachse und Faltungsergebnis gleichzeitig zurückgibt. Wird das Ergebnis einem Skalar zugewiesen, soll nur das Faltungsergebnis zurückgegeben werden.

4.2. Testen Sie die Funktion anhand der Aufgabe 2.4 der 5. Übung zu »Deterministische Signale und Systeme«. Verwenden Sie dabei einen Kondensator von 1 µF und einen Widerstand von 1 MΩ.

# Lösung

4.1. Um es sich einfach zu machen, implementiert man die beiden Funktionen natürlich nicht neu, sondern benutzt sie einfach so, wie sie zu Beginn dieser Aufgabe geschrieben wurden. Man weist nur die Rückgabewerte entsprechend der Aufgabenstellung zu (`tconv.m`):

```
function [f, t] = tconv(t1,f1,t2,f2)
% TCONV calculates time axis and convolution for two input
%   signals or systems

t = ctime(t1, f1, t2, f2);
f = cval(t1, f1, t2, f2);
```

4.2. Die Impulsantwort des Systems aus der Übung zu Deterministische Signale und Systeme hat im Bereich $0 \leq t \leq 1$ die Impulsantwort $1-t$ und ist sonst 0, woraus sich die Zeitachse und die diskretisierte Impulsantwort

```
t1 = 0 : .001 : 1;
h1 = 1-t1;
```

ergeben. Dabei wurden kleine Zeitschritte gewählt, um eine gute Annäherung an die algebraische Lösung zu erhalten.

Mit den gegebenen Bauteilewerten

```
R = 1e6;
C = 1e-6;
```

ergibt sich die Impulsantwort des zweiten Systems (samt Zeitachse) zu

```
t2 = 0 : .001 : 5;
h2 = 1/(R*C) * exp(-1/(R*C)*t2);
```

Nun nur noch falten

```
[h, t] = tconv(t1, h1, t2, h2);
```

und plotten

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

und das Ergebnis kann mit der Lösung der Aufgabe aus Deterministische Signale und Systeme verglichen werden.