v1.2

Matlab-Übungen zu Deterministische Signale und Systeme Musterlösung zur 3. Übung

1. Aufgabe

Die Fouriertransformation ist durch die Beziehung

$$X(j\omega) = \int_{-\infty}^{\infty} x(t)\, e^{-j\omega t}\, dt \tag{1}$$

gegeben. Diese Gleichung lässt sich so nicht direkt in ein Matlab-Programm umsetzen, da eine unendliche Zeitachse auf eine unendliche Frequenzachse abgebildet wird und der Rechner naturgemäß seine Probleme mit solch großen Zahlen wie ∞ hat. Um dieses Problem zu umgehen, beschränken wir uns auf Funktionen endlicher Amplitude und endlicher zeitlicher Ausdehnung (also auf Impulse).

Ein weiteres Problem ist, dass die Funktionen nur durch eine endliche Anzahl von Stützstellen dargestellt werden können.

Eine implementierbare Form von Gleichung (1) hat also die Form

$$X(j\omega_n) = \int_{T_1}^{T_2} x(t)\, e^{-j\omega_n t}\, dt \tag{2}$$

wobei ω_n nun eine endliche Anzahl von Stützstellen $\omega_1,\ \omega_2,\ \ldots,\ \omega_N$ repräsentiert.

1.1. Das Integral in Gleichung (2) muss in geeigneter Weise angenähert werden, da nur Funktionswerte von $x(t)$ an endlich vielen Stützstellen vorhanden sind. Matlab stellt hierfür u. a. die Funktion `trapz` bereit, die ein Integral mit der Trapezmethode löst. Zuerst werden wir uns die Funktionsweise dieser Funktion an einem Beispiel verdeutlichen.

(1.1.1) Stellen Sie die Zahlendarstellung von Matlab durch Eingabe von `format short eng` um. Wie das entsprechende Zahlenformat aufgebaut ist und welche weiteren Eingaben `format` verarbeitet, entnehmen Sie bitte den entsprechenden Hilfeseiten.

Erzeugen Sie nun einen Spaltenvektor `t`, der eine »Zeitachse« im Intervall $[0, 2\pi]$ in Schritten von $\pi/180$ enthält.

(1.1.2) Erzeugen Sie eine Matrix, die aus drei Spaltenvektoren besteht, die die Funktionen $\sin(t)$, e^{jt} und $1(t)$ enthält.

M. Sc. M. Trinh Hoang (S108/207); K. Cui
Prof. Dr.-Ing. A. Klein
Prof. Dr.-Ing. M. Pesavento
Wintersemester 2018/19

int

(1.1.3) Bestimmen Sie das Integral der drei Funktionen über den Zeitbereich in `t` mithilfe der Funktion `trapz`.

Schreiben Sie eine Funktion `ftrans`, die eine Zeitachse als Spaltenvektor, ein Signal als Zeilenvektor und eine Frequenzachse als Zeilenvektor übergeben bekommt und die Fouriertransformierte des Signals an den übergebenen Frequenzpunkten zurückliefert. Benutzen Sie zur Integration die Funktion `trapz`.

Legen Sie zunächst, um die Funktion testen zu können, einen Zeitachsenvektor `t` an, der das Intervall $[0, 1\,\mathrm{s})$ mit einer Auflösung von 0,01 s enthält (`t` muss ein Spaltenvektor sein). Erzeugen Sie weiterhin einen Zeilenvektor, der die entsprechenden Signalwerte eines Rechteckimpulses der Amplitude 1 und Länge 0,5 s in der Mitte des gegebenen Zeitbereichs enthält. Erzeugen Sie als drittes noch den Vektor für die Frequenzachse mit $-100 \leq \omega \leq 100$ und einer Auflösung von 0,1.

Plotten Sie Betrag und Phase der Transformierten des Rechteckimpulses. Entspricht das Ergebnis der erwarteten (analytischen) Lösung?

1.3. Wir haben aus Gründen der Implementierbarkeit festgelegt, dass wir uns auf ein Signal endlicher zeitlicher Ausdehnung beschränken. Ein zeitbegrenztes Signal ist im (Fourier-) Frequenzbereich aber immer unendlich breit, wodurch bei der Berechnung eines eingeschränkten Frequenzbereichs ein Fehler entsteht. Diese Energiedifferenz soll nun bestimmt werden.

Der Satz von Parseval besagt, dass die Energie eines Impulses im Zeit- und im Frequenzbereich gleich groß sein muss. Als Gleichung ausgedrückt:

$$\int_{-\infty}^{\infty} |x(t)|^2 \, dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |X(j\omega)|^2 \, d\omega. \tag{3}$$

Berechnen Sie die Abweichung der Energie des Impulses und seiner Transformierten aus Aufgabe 1.2. Welche Möglichkeiten haben Sie, diese Differenz zu veringern?

1.4. Überprüfen Sie die Lösungen aus den Aufgaben 1–3 der 3. DSS-Übung unter Verwendung Ihrer Transformationsfunktion.

Lösung

1.1. Die Berechnung der Integrale erledigt das folgende Stück Quelltext:

```
format short eng;

t = (0 : pi/180 : 2*pi).';
x = [sin(t) exp(j*t) ones(length(t),1)];

trapz(t, x);
```

1.2. Eine einfache Variante der Funktion `ftrans` wäre

```
function X = ftrans(t, x, w)
% FTRANS   Fourier transformation
%
%   ftrans(t,x,w) returns the Fourier transformation of x at
%     the frequency points w with respect to the given time
%     axis.
%
%     Some parameters have to fulfill special conditions:
%
%       t: column vector
%       x: vector
%       w: row vector
%
%     t and x have to hold the same number of elements .

X = trapz(t, diag(x)*exp(-j*t*w));

end
```

`diag(x)` erzeugt dabei eine Diagonalmatrix aus dem Vektor `x`. Multipliziert man diese auf die Matrix, die man mit `t*w` aufspannt, wird jedes Element einer Zeile mit dem entsprechenden x-Wert multipliziert (also die Elemente aus Zeile 1 mit `x(1)`, die Elemente aus Zeile 2 mit `x(1)` usw.). Die entsprechende Matrix hat die Form

$$\begin{pmatrix} x(T_1)\cdot e^{-jT_1\omega_1} & x(T_1)\cdot e^{-jT_1\omega_2} & \cdots & x(T_1)\cdot e^{-jT_1\omega_N} \\ x(T_1+\Delta t)\cdot e^{-j(T_1+\Delta t)\omega_1} & & & \\ \vdots & & \ddots & \\ x(T_2)\cdot e^{-jT_2\omega_1} & & & x(T_2)\cdot e^{-jT_2\omega_N} \end{pmatrix}$$

wobei ω_1 und ω_N Unter- und Obergrenze des Frequenzbereichs sind und Δt den Abstand zwischen zwei Stützstellen im Zeitbereich darstellt. Wie schon im Aufgabenteil 1.1 gezeigt integriert `trapz` spaltenweise, und durch die Form dieser Matrix läßt sich nun in einem Schritt das Integral für alle Frequenzen lösen. Natürlich ist auch eine Lösung mittels `for`-Schleifen möglich.

Es ist aber generell zu empfehlen, einige Überprüfungen der Eingabewerte durchzuführen. Weiterhin ist es natürlich möglich und in diesem Fall auch ratsam, die Einschränkung auf Spalten- oder Zeilenvektoren bestimmter Parameter aufzuheben und sich selbst um die benötigte Form zu kümmern. Eine entsprechende Variante der Funktion `ftrans` könnte wie folgt aussehen:

```
function X = ftrans2(t, x, w)
% FTRANS   Fourier transformation
%
%   ftrans(t, x, w) returns the Fourier transformation of x at
%     the frequency points w with respect to the given time
%     axis. t and x have to hold the same number of elements.
```

```
% Validation of input values
if ~isvector(t) || ~isvector(x) || ~isvector(w)
    error('t, x and w have to be vectors!');
end

if length(t) ~= length(x)
    error('t and x have to have the same length!');
end

% Conversion of vectors
[a, b] = size(t);
if a == 1
    t = t.';
end

[a, b] = size(w);
if b == 1
    w = w.';
end

% Calculation of Fourier transformation
X = trapz(t, diag(x)*exp(-j*t*w));

end
```

Die Konversion der Vektoren `t` und `w` kann man sich sparen, wenn man

```
X = trapz(t, diag(x)*exp(-j*t(:)*w(:).'));
```

oder

```
X = trapz(t, exp(-j*w(:)*t(:).')*diag(x), 2);
```

verwendet. Der Parameter `2` bei der Trapezintegration in der zweiten Variante bedeutet, dass über die zweite Dimension der Matrix, d.h. die Zeilen integriert wird. Das Ergebnis ist folglich ein Spaltenvektor.

Nach Festlegen der Zeitachse

```
t = (0 : .01 : .99).';
```

des Rechteckimpulses

```
x = [ zeros(1,length(t)/4) ones(1,length(t)/4) ...
      ones(1,length(t)/4) zeros(1,length(t)/4) ];
```

und der Frequenzachse

```
w = -100 : .1 : 100;
```

bestimmt man mit

```
X = ftrans(t, x, w);
```

die Fouriertransformation von $x(t)$.

Mittels des `plotyy`-Befehls stellt man Betrag und Phase in einmen Diagramm dar:

```
figure;
plotyy(w, abs(X), w, phase(X)); grid;
xlabel('\omega');
ylabel('\color{blue}|x|␣␣␣\color{darkGreen}\phi(x)');
```

Dabei darf natürlich eine geeignete Achsenbeschriftung nicht fehlen! Für den Betrag sehen wir wie erwartet den Betrag der si Funktion. Die Phase ist linear, weil die Rechteckfuntion verschoben ist und hat Sprünge um π wo die si Funktion ihr Vorzeichen wechselt.

1.3. Aus den Variablen `x` und `X` lässt sich leicht die Energieabweichung berechnen, wobei auch wieder die Trapezintegration benutzt wird:

```
pa = trapz(t, abs(x).^2); %Berechnung Zeitbereich
pb = trapz(w, abs(X).^2)/2/pi; %Berechnung Frequenzbereich
err = pa - pb
```

Die Differenz läßt sich nur verringern, indem ein vergrößerter Frequenzbereich verwendet wird.

1.4. Das Verfahren wird hier am Beispiel einer der drei Aufgaben, Aufgabe 1, erläutert. Die anderen Aufgaben lassen sich analog dazu lösen.

Nach dem Anlegen der Zeitachse

```
t2 = (-10 : .04 : 10).'; % Zeitachse
```

wird die Parabel erzeugt, aus der die 5 Turmspitzen bestehen. Dazu wird zunächst eine eigene Zeitachse für diese Parabel

```
tp2 = (-1 : .04 : 1).'; % Zeitachse fuer Parabel
```

und schließlich die Parabel selbst erzeugt:

```
p2 = -tp2.^2 + 1;      % Parabel
```

Die Funktion $x(t)$ wird nun abschnittsweise aus Konstanten und den Parabeln »zusammengebastelt«:

```
x2 = zeros(1, length(t2)); % Zusammenbau des "Turms"
x2(find(t2>=-5 & t2<=-3)) = 5 + p2;
x2(find(t2>=-3 & t2<=-1)) = 6 + p2;
x2(find(t2>=-1 & t2<=1)) = 7 + p2;
x2(find(t2>=1  & t2<=3)) = 6 + p2;
x2(find(t2>=3  & t2<=5)) = 5 + p2;
```

Schließlich wird noch die Frequenzachse definiert:

```
w2 = -25 : .1 : 25;
```

Die Transformation selbst funktioniert analog zu den vorangegangenen Aufgabenteilen:

```
X2 = ftrans2(t2, x2, w2);
```

Als Referenz wird die Lösung aus der DSS-Aufgabe in einem weiteren Vektor gespeichert.

```
Y2 = 4./(w2.^2) .* (sinc(w2/pi)-cos(w2)) ...
     .* (1+2*cos(4*w2)+2*cos(2*w2)) ...
     + 50*sinc(5*w2/pi)+6*sinc(3*w2/pi)+2*sinc(w2/pi);
```

Plottet man nun die Beträge der beiden Vektoren über dem Frequenzvektor

```
figure;
plot(w2, abs(X2), w2, abs(Y2)); grid;
xlabel('\omega');
ylabel('|X_2|, |Y_2|');
legend('X_2','Y_2');
```

sieht man nur eine einzelne Kurve, die beiden Lösungen liegen übereinander.

Betrachtet man die Phase, sieht man zwar eine ähnliche Struktur, allerdings sind Differenzen von $n \cdot 2\pi$ mit ganzzahligem n zu beobachten:

```
figure;
plot(w2, phase(X2), w2, phase(Y2)); grid;
xlabel('\omega');
ylabel('\phi(X_2), \phi(Y_2)');
legend('X_2','Y_2');
```

Aufgrund der Phase der tatsächlichen Lösung (Y_2) würde man erwarten, dass die Lösung rein reell ist, die Sprünge zwischen 0 und π drücken aus, dass in der Betragsdarstellung zu sehenden »Nebenkeulen« eben in in die negative Richtung der x-Achse zeigen. Das gleiche drücken aber auch die um $n \cdot 2\pi$ verschobenen Phasenwerte aus. Doch woher kommt nun die Abweichung?

Betrachtet man den Imaginärteil der beiden Funktionen

```
figure;
plot(w2, imag(X2), w2, imag(Y2)); grid;
xlabel('\omega');
ylabel('\Im(X_2), \Im(Y_2)');
legend('X_2', 'Y_2');
```

erkennt man, dass die mit Matlab berechnete Lösung einen Fehler aufweist, der Imaginärteil ist nicht null. Die Hilfe zur Funktion `phase` erklärt, dass diese Funktion versucht, die Phase kontinuierlich weiterzuführen, auch über die Grenze von $\pm\pi$ hinweg. Hier sorgt nun der Fehler dafür, dass `phase` die Phasenwerte »falsch« zuordnet.

2. Aufgabe

Der einfachste Ansatz zur numerischen Differentiation ist der Übergang »zurück« zum Differenzenquotienten. Dabei gibt es mehrere Möglichkeiten, diesen zu bilden.

Der einfachste Ansatz ist das Bilden des Vorwärtsdifferenzenquotienten gemäß

$$f'(x_0) \approx \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}, \tag{4}$$

wobei Δx genau dem Abstand zwischen zwei Abtastpunkten entspricht.

Eine weitere Möglichkeit ist das Bilden des zentralen Differentenquotienten gemäß

$$f'(x_0) \approx \frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x}. \tag{5}$$

2.1. Speichern Sie die Werte für $-5 \leq x \leq 5$ mit einem Abstand von 0,1 aufsteigend in einem Zeilenvektor `x`. Speichern Sie anschließend die Funktion $f(x) = x^2$ in einem Zeilenvektor `f`. Stellen Sie mit den gespeicherten Variablen die Funktion $f(x)$ graphisch dar.

2.2. Schreiben Sie die Funktion `vdiffquot`, die die Parameter `x` und `f` übergeben bekommt und als Rückgabewert die Ableitung von `f`, berechnet nach Gleichung 4, zurückgibt.

2.3. Schreiben Sie die Funktion `zdiffquot`, die auch die Parameter `x` und `f` übergeben bekommt und als Rückgabewert die Ableitung von `f`, berechnet nach Gleichung 5, zurückgibt.

2.4. Wenden Sie nacheinander die Funktionen `vdiffquot` und `zdiffquot` auf die Funktion $f(x)$ an und stellen Sie das jeweilige Ergebnis gemeinsam mit $f(x)$ in einem Diagramm dar.

2.5. Berechnen Sie den Betrag des Fehlerquadrates $e_v^2(x) = \left|\frac{d}{dx}f(x) - f_v'(x)\right|^2$, wobei $f_v'(x)$ die in 2.2 berechnete Ableitung meint. Berechnen Sie nach dem gleichen Schema auch den Fehler des zentralen Differentenquotienten gegenüber den tatsächlichen Werten. Stellen Sie die beiden Fehler logarithmisch dar, indem sie zum Plotten die Funktion `semilogy` verwenden. Haben Sie eine Idee worauf die Unterschiede in den beiden Fehlern beruhen?

Lösung

2.1. Der entsprechende Code sieht wie folgt aus:

```
x = -5 : .1 : 5;
f = x.^2;

figure;
plot(x, f);
```

2.2. Eine Möglichkeit, das zu implementieren, ist die Folgende:

```
function out = vdiffquot(x, f)

% Ueberpruefen, ob die Vektoren die gleiche Laenge haben. Besser
% noch waere ein Pruefen, ob beides auch Vektoren sind.
if length(x) ~= length(f)
    error('Vector␣x␣and␣f␣have␣to␣have␣the␣same␣length!')
```

```
end

out = (f(2:end) - f(1:end-1)) ./ (x(2:end) - x(1:end-1));

end
```

Diese Art der Implementation zeigt einen Vorteil, den man aufgrund der Möglichkeit, in Matlab mit einfacher Syntax mit Vektoren und Matrizen zu rechnen, hat: Man kann oftmals sehr einfachen Code schreiben, der ohne Schleifen auskommt.

Die Implementation ist dabei so flexibel, dass sogar Funktionen mit ungleichmäßigen Abtastabständen verwendet werden können (das entspräche allerdings nicht mehr der Definition aus Gleichung 4 bzw. 5).

2.3. Analog implementiert man dann `zdiffquot.m`:

```
function out = zdiffquot(x, f)

% Ueberpruefen, ob die Vektoren die gleiche Laenge haben. Besser
% noch waere ein Pruefen, ob beides auch Vektoren sind.
if length(x) ~= length(f)
    error('Vector x and f have to have the same length!')
end

out = (f(3:end) - f(1:end-2)) ./ (x(3:end) - x(1:end-2));

end
```

2.4. Der entsprechende Codeabschnitt sieht so aus:

```
y = vdiffquot(x, f);
z = zdiffquot(x, f);

figure;
plot(x, f, x(1:end-1), y, x(2:end-1), z);
% Hier muss beachtet werden, dass die Ergebnisse eine andere
% Laenge als die Eingabevektoren haben.

legend('f(x)', 'Ableitung mit Vorwaertsdifferenzenquotient', ...
       'Ableitung mit zentralem Differenzenquotienten');
```

2.5. Mit

```
d = 2*x; % Das ist die tatsaechliche Loesung
figure
semilogy(x(1:end-1), abs(d(1:end-1)-y).^2, ...
         x(2:end-1), abs(d(2:end-1)-z).^2);
legend('Fehler Vorwaertsdifferenzenquotient', ...
       'Fehler zentraler Differenzenquotient');
```

erreicht man das gewünschte.

Der quadratische Fehler für den Vorwärtsdifferenzenquotienten beträgt konstant 0,01. Der qudratische Fehler des zentralen Differenzenquotienten ist kleiner als 8e-29. Die Lücken im Fehler des zentralen Differenzenquotienten kommen von Fehlerwerten, die (im Rahmen der Rechengenauigkeit) exakt den Wert 0 annehmen, welcher im logarithmischen Diagramm nicht dargestellt werden kann.

Wir können den numerische Fehler näher Beleuchten, wenn wir die Approximationsordnung der numerischen Ableitungen betrachten.

Die Taylor-Entwicklung einer Funktion ist

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \; + \; \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1} \tag{6}$$

wobei der rechte Summand das Restglied mit ξ zwischne x_0 und x darstellt. Wenn wir $x - x_0 = \Delta x$ setzen können wir die Entwicklung bis zur ersten Ordnung als

$$f(x_0 + \Delta x) = f(x_0) \; + \; f'(x_0)\Delta x \; + \; \mathcal{O}(\Delta x^2) \tag{7}$$

schreiben. Eingesetzt in Gleichung 4 ergibt für den rechtsseitgen Differenzenquotienten:

$$\frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} = \frac{f'(x_0)\Delta x \; + \; \mathcal{O}(\Delta x^2)}{\Delta x} \tag{8}$$

$$= f'(x_0) \; + \; \mathcal{O}(\Delta x) \tag{9}$$

Das heißt der einsitige Differenzenquotient approximiert die Ableitung in 1. Ordnung. Der Fehler läßt sich durch das Restglied abschätzen und ist für die Funktion $f(x) = x^2$

$$\frac{f^{(2)}(\xi)}{2}\Delta x = \frac{2}{2}\Delta x = \Delta x. \tag{10}$$

Da unser Abstand $\Delta x = 0{,}1$ gewählt wurde, ist der quadratische Fehler erwartungsgemäß 0,01.

Wenn wir die Taylor-Entwicklung bis zur 2. Ordnung schreiben

$$f(x_0 \pm \Delta x) = f(x_0) \; \pm \; f'(x_0)\Delta x \; + \; \frac{f''(x_0)}{2}\Delta x^2 \; \pm \; \mathcal{O}(\Delta x^3) \tag{11}$$

und in Gleichung 5 einsetzen, erhalten wir für den zentralen Differenzenquotienten:

$$\frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x} = \frac{2f'(x_0)\Delta x \; + \; \mathcal{O}(\Delta x^3)}{2\Delta x} \tag{12}$$

$$= f'(x_0) \; + \; \mathcal{O}(\Delta x^2) \tag{13}$$

Der symmetrische Differenzenquotient approximiert die Ableitung in 2. Ordnung, d. h. die Ableitung einer Parabel ist numerisch exakt. Die absoluten (nicht quadrierten) Fehler zwischen 0 und 1e-15 resultieren aus der Rechengenauigkeit von Matlab bzw. allgemein auf der Genauigkeit einer 64 bit double precision Gleitkommazahl (siehe IEEE 754).

3. Aufgabe

3.1. Schreiben Sie eine Funktion `fakultaet`, die die Fakultät einer übergebenen Zahl berechnet. Arbeiten Sie dabei mit dem Prinzip der Rekursion (`fakultaet` ruft sich selbst auf). Überprüfen Sie die korrekte Arbeitsweise der Funktion mit Werten von denen Sie auch von Hand noch die Fakultät berechnen können, etwa 3!, 4!, 5! und 6!.

3.2. Berechnen Sie mit dieser Funktion die Fakultät von 10, 23, 42, 99 und 101.

Lösung

3.1. Um die Funktion rekursiv zu implementieren, zerlegt man das Bilden der Fakultät gemäß

$$
\begin{aligned}
N! &= N \cdot (N-1)! && (14)\\
&= N \cdot \Big[(N-1) \cdot (N-2)!\Big]\\
&= N \cdot \Big[(N-1) \cdot \Big[(N-2) \cdot (N-3)!\Big]\Big]\\
&\;\;\vdots\\
&= N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot 2 \cdot 1 && (15)
\end{aligned}
$$

Hier kann man direkt ablesen, wie die Rekursion zu bilden ist (Gleichung 14). Auch die Abbruchbedingung ist in Gleichung 15 direkt ablesbar.

Zuerst überprüfen wir, ob es sich bei dem Eingabewert um eine skalare ganze Zahl handelt, die nicht negativ ist:

```
function out = fakultaet(in)
% FAKULTAET   calculates the factorial.
%
%       in      integer input
%
%   out     fakultaet(in) = 1 * 2 * 3 * ... * (in-1) * in
%

if ~isscalar(in) || in~=floor(in) || in < 0
    error('Factorial is only defined for scalar non-negative integers!');
end
```

Mit

```
if in == 0 || in == 1
    out = 1;
```

fangen wir die Abbruchbedingung ab und berücksichtigen gleichzeitig den Fall $0! = 1$. Ist die Abbruchbedingung nicht erreicht, ruft die Funktion sich selbst auf, mit einem um eins verminderten Übergabewert, und multipliziert den eigenen Übergabewert an das Ergebnis des Funktionsaufrufs:

```
else
    out = in * fakultaet(in-1);
end

end
```

3.2. Mit den Zeilen

```
for k = [10 23 42 99]
    disp([num2str(k) '! = ' num2str(fakultaet(k))]);
end
```

gibt man die Lösungen auf dem Bildschirm aus. Neben der etwas »schöneren« Darstellung der Ergebnisausgabe hat das den angenehmen Nebeneffekt, dass die Zahlen vollständig (und nicht in Exponentialschreibweise) ausgegeben werden.

Der Versuch, die Fakultät von 501 zu berechnen, mündet in der folgenden Fehlermeldung:

```
Maximum recursion limit of 500 reached. ...
```

Dieser Wert lässt sich zwar erhöhen, allerdings muss man gut aufpassen, wenn man rekursive Funktionen in Matlab implementiert, um nicht in Endlosschleifen zu geraten oder die Rechenzeit unnötig zu verlängern.