Korrekt normierte FFT eines Zeitsignals $A(t)$ und analytische Bildung des Spektrums $FFT(\frac{d}{dt}A(t))$ (mittels MATLAB)

Jan-Philip Gehrcke, 5. März 2009

1 Abstract

Bei der Untersuchung des Frequenzspektrums eines Zeitsignals ist es oftmals von Interesse, nicht nur die Frequenzachse richtig zu eichen, sondern auch die Amplitudenachse. Der (MATLAB-)Weg von der Bildung eines Zeitsignals bis zu einem Fourierspektrum, anhanddessen man die Amplituden korrekt ablesen kann, wird mittels eines einfachen Beispiels erläutert. Es stellt sich heraus, dass die MATLAB-Hilfe hier in die Irre führt.

Eine weitere Problematik ist die oftmals kritische numerische Bildung der Ableitung eines Zeitsignals. Interessiert die Fouriertransformierte (FT) der Zeitableitung, so lässt sich dieses Spektrum analytisch aus der FT des Zeitsignals gewinnen, sodass die numerische Ableitungsbildung erspart bleibt. Dieses wird ebenfalls anhand eines Beispiels gezeigt.

2 Spektrale Untersuchung eines Zeitsignals $A(t)$

Das zu untersuchende Zeitsignal $A(t)$ soll aus einem Offset und zwei harmonischen Frequenzanteilen bestehen:

$$A(t) = c + a_1 \sin(\omega_1 t) + a_2 \sin(\omega_2 t) \tag{1}$$

Der Offset soll $c = 3$ betragen, die Amplituden $a_1 = 1$ und $a_2 = 1$. Die Frequenzen werden zu $\omega_1 = 2\pi \cdot 1\,Hz$ und $\omega_2 = 2\pi \cdot 2\,Hz$ gewählt. Es folgt MATLAB-Code zur Definition dieser Konstanten:

```
a1 = 1;
f1 = 1;
a2 = 1;
f2 = 2;
w1 = 2 * pi * f1;
w2 = 2 * pi * f2;
c = 3;
```

2.1 FFT-Vorbereitung: Wahl von Samplingrate und anderen Konstanten

Für die Fouriertransformation muss das Zeitsignal mit der Samplingrate f_{sample} abgetastet werden. Das 500-fache der höchsten vorkommenden Frequenz ist bereits ein sehr hoher Wert.

```
f_sample = 500 * f2;
```

Die Zeit L gibt an, wie lange das Signal abgetastet werden soll. Hier werden 10000 Perioden der niedrigsten Frequenz gewählt; ebenfalls ein sehr hoher Wert.

```
L = 10000/f1;
```

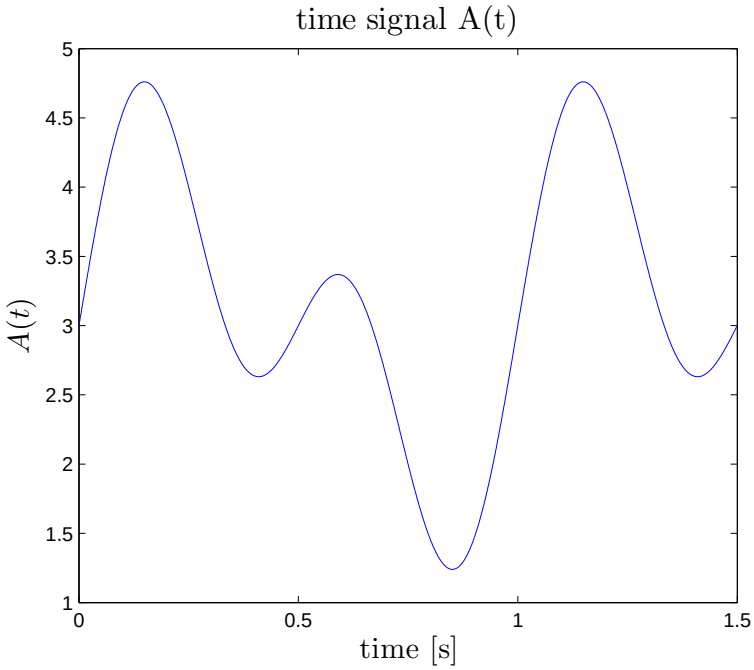

Abb. 1: Zeitsignal $A(t)$. Harmonische Beiträge: Amplitude 1/Frequenz $1\,Hz$; Amplitude 1/Frequenz $2\,Hz$. Konstanter Beitrag: Offset mit Höhe 3.

Jetzt sollen alle Zeitpunkte, zu denen ein Datenpunkt genommen wird, in einen MATLAB-Vektor t_{vec} geschrieben werden:

```
Ts = 1/f_sample;
t_vec = (0:Ts:L);
```

Dieser „Timestampvektor“ wird nun als Argument verwendet, um das abgetastete Zeitsignal $A(t)$ zu erhalten:

```
A = c + a1 .* sin(w1 .* t_vec) + a2 .* sin(w2 .* t_vec);
```

$A(t)$ ist in Abb. 1 dargestellt.

Erhält eine gewöhnliche DFT (Diskrete Fouriertransformation) N Datenpunkte eines Zeitsignals zur Auswertung, so gibt sie auch N Datenpunkte zurück, die „Amplituden“. Zur Darstellung Amplitude-über-Frequenz werden entsprechend ebenfalls N Frequenzwerte benötigt. Die maximal detektierbare Frequenz bei der DFT ist die halbe Abtastrate $f_{sample}/2$. Das von einer FT produzierte Spektrum ist symmetrisch um 0. Im Falle einer DFT (z.B. FFT) geht das Frequenzspektrum von $-f_{sample}/2$ bis $f_{sample}/2$. MATLAB's FFT-Funktion jedoch ordnet das Spektrum symmetrisch um $f_{sample}/2$ an, sodass der Frequenzbereich von $0\,Hz$ bis f_{sample} reicht. Dieser Frequenzbereich muss für die Darstellung Amplitude-über-Frequenz in N Werte zerlegt werden. Dann bildet der i-te Datenpunkt der DFT-Ausgabe ein Wertepaar mit dem i-ten Frequenzwert. Im folgenden MATLAB-Code heißt N *samplepoints*. Die Zahl der Datenpunkte entspricht der Zahl der abgetasteten Zeitpunkte. `linspace(start,ende,anzahl)` erzeugt einen MATLAB-Vektor von `start` bis `ende` mit genau `anzahl` Elementen.

```
samplepoints = length(t_vec);
freqs = linspace(0,f_sample,samplepoints);
```

2.2 Korrekt normierte FFT

2.2.1 Fehlerquellen

Es ist das Ziel, die Amplituden der zum Gesamtsignal beitragenden Schwingungen möglichst korrekt im Spektrum in Form von Peakhöhen ablesen zu können. Die Abweichung der Peakhöhen von den realen Amplituden hängt von zwei Faktoren ab. Einerseits das Ergebnis einer FFT normiert werden, wobei die Art der Normierung vom speziellen Algorithmus abhängt. Diese Normierung kann man - vorausgesetzt man kennt sie - immer korrekt implementieren und somit als Fehlerquelle ausschließen.

Die zweite verbleibende Fehlerquelle ist vorgegeben durch die Länge des für die Fouriertransformation verwendeten Signals, die Samplingrate und die Fensterung des Signals. Länge und Samplingrate definieren zusammen die Frequenzauflösung des entstehenden Spektrums nach Transformation: $\Delta f = f_{sample}/N$. Je schlechter die Frequenzauflösung ist, desto fehlerbehafteter sind die Amplituden im Spektrum.

Es sollte immer darauf geachtet werden, dass genau n Perioden des Signals zur Transformation verwendet werden. Wird durch ungeschickte Fensterung (Wahl eines Zeitintervalls, in dem Daten für die Transformation akquiriert werden) eine nicht-ganzzahlige Zahl Perioden aufgenommen, verbreitert dies die Peaks und verringert ihre Höhe. Dieser Effekt tritt bei kleinen zur Transformation verwendeten Signallängen verstärkt auf und wird bei sehr großen Längen vernachlässigbar. Durch technische Grenzen ist oftmals eine relativ kurze Datenakquisitionszeit pro Transformation vorgegeben. In diesen Fällen ist also auf eine günstige Fensterung zu achten.

Im vorliegenden Beispiel ist durch hohe Samplingrate und Signallänge eine sehr hohe Frequenzauflösung gewählt, sodass die zweite Fehlerkategorie minimiert ist. Es kann also untersucht werden, welche die korrekte Normierung ist.

2.2.2 Matlab-Hilfe fehlerhaft

Der Blick in die MATLAB-Hilfe zu `fft()`, um die korrekte Normierung herauszufinden, war leider enttäuschend und brachte einige Verwirrung. Das dort vorhandene Beispiel ist - meiner Meinung nach - stark fehlerhaft:

```
% AUS DER MATLAB–HILFE zu fft()
% verwendete Variablennamen stehen daher in keinem Kontext zu diesem Dokument!
NFFT = 2^nextpow2(L); % Next power of 2 from length of y
Y = fft(y,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2);
plot(f,2*abs(Y(1:NFFT/2)))
```

`y` ist das Zeitsignal, bestehend aus zwei harmonischen Beiträgen (Amplituden 0.7 und 1) sowie einem starken Rauschen (maximal 2). `L` ist die **Anzahl** der Datenpunkte. Der Autor dieses MATLAB-Hilfe-Eintrags wollte vorbildlich handeln und hat die Anzahl der für die FFT verwendeten Datenpunkte gleich einer Zweierpotenz (`NFFT`) gewählt, um den FFT-Algorithmus möglichst schnell zu machen. Er hat aber `L` willkürlich gewählt; also kann - bei Übernahme seines Beispiels - `NFFT` teilweise deutlich größer als `L` werden. Ist das zweite Argument von `fft()` größer als die Länge des ersten Argumentes, füllt MATLAB den Vektor im ersten Argument mit Nullen auf, bis dieser genauso lang ist, wie das zweite Argument vorgibt. Wenn der Abstand zur nächsten Zweierpotenz groß ist, kann dieses Verfahren das Zeitsignal deutlich verfälschen, indem es über einen großen Zeitraum „Null“ zu sein scheint!

Das Ergebnis der FFT, die komplexe Fouriertransformierte `Y`, normiert der Autor auf `L`. Eine DFT wird je nach Algorithmus - jedoch immer auf $N/2$, N oder $\sqrt{N}$ normiert, wenn N die Zahl der übergebenen Datenpunkte ist. Im Falle der MATLAB-FFT muss auf N normiert werden, wie weiter unten erläutert wird. Der Autor hätte also konsequenter Weise durch `NFFT` dividieren müssen, weil er die FFT mit `NFFT` Datenpunkten durchführt. Durch diesen Normierungsfehler werden die Amplituden willkürlich um den Faktor `NFFT/L` verändert. Der Autor hat Glück, dass

sein gewähltes `L` = 1000 nahe der nächsten Zweierpotenz 1024 liegt. Im Extremfall kann aber ein Fehlfaktor von nahezu 2 entstehen (z.B. bei `L`= 1025). Die selbe Betrachtung gilt für das scheinbare „Null-Signal“. Im Extremfall ist die zweite Hälfte des ausgewerteten Zeitsignals ungewollt 0, was sämtliche Frequenzinformation sehr stark verfälscht.

Im Beispiel der MATLABwird dann `2*abs(Y)` über dem Frequenzvektor `f` aufgetragen. Dies ist korrekt und wird in Teil 2.2.3 näher erläutert. Im Ergebnis des Beispiels weichen die Amplituden im gezeichneten Spektrum von 0.7 und 1 ab. Erklärung des Autors: „The main reason the amplitudes are not exactly at 0.7 and 1 is because of the noise.“ Durch den glücklichen Zufall, mit `L`= 1000 nahe 1024 zu liegen, ist diese Begründung sogar richtig. Wenn Nutzer jedoch diese Methode für Ihr Problem übernehmen, wird die Hauptfehlerursache - wie bereits begründet - in den meisten Fällen in der falschen Normierung und der Verfälschung des Zeitsignals durch angehängte Nullen zu finden sein.

Es bleibt anzumerken, dass die Wahl von Zweierpotenzvektoren zwar geschickt klingt, jedoch praktisch nicht nötig ist. Ebenfalls in der MATLAB-Hilfe steht geschrieben (auf einer anderen Seite, welche allgemein zu FFT in MATLAB informiert):

„The execution time of an FFT algorithm depends on the transform length. It is fastest when the transform length is a power of two, and almost as fast when the transform length has only small prime factors. It is typically slower for transform lengths that are prime or have large prime factors. **Time differences, however, are reduced to insignificance by modern FFT algorithms such as those used in MATLAB. Adjusting the transform length for efficiency is usually unnecessary in practice.**“

Dies liegt daran, dass die Komplexität der DFT durch moderne FFT-Algorithmen, wie sie in MATLAB verwendet sind, immer $O(N \log(N))$ ist bei Datensätzen beliebiger Größe N und nicht nur bei Datensätzen mit $N = 2^x$.

Die Zweierpotenz-Problematik kann man also i.A. einfach umgehen, ohne sich Sorgen um die Ausführungsgeschwindigkeit machen zu müssen.

2.2.3 Die Faktoren $\frac{1}{N}$ und 2

Es wird wieder das Beispiel von $A(t)$ betrachtet. Schritt für Schritt soll nun ein korrektes Amplitudenspektrum von $A(t)$ erzeugt werden. Begonnen wird mit der Erzeugung der komplexen Fouriertransformierten `fft_A`. Es wird aus genannten Gründen keine Anstrengung unternommen, einen Vektor der Größe einer Zweierpotenz zu transformieren:

```
fft_A = fft(A);
```

In der experimentellen Auswertung dieser komplexen Größe interessieren meist Betrag und Phase, wobei hier nur der Betrag betrachtet wird, welcher die Amplituden bestimmt:

```
freq_spectrum_A = abs(fft_A);
```

Nun fehlen bis zum korrekten Amplitudenspektrum noch die Faktoren 2 und $1/samplepoints$:

```
freq_spectrum_A = 2 * freq_spectrum_A / samplepoints;
```

Diese beiden Faktoren werden nun erläutert. Dass bei der in MATLAB implementierten FFT („FFTW“) auf N normiert werden muss, erfährt man indirekt auf der Hilfe-Seite zu `fft()` an der Stelle „The functions X = fft(x) and x = ifft(X) implement the transform and inverse transform pair given for vectors of length by:“.

Hier sind sowohl `fft()` als auch `ifft()`, also die inverse FFT, mathematisch formuliert. Bei der Inversen taucht der Faktor $1/N$ auf, sodass `ifft(fft(A))` wieder `A` ergeben würde. Daher muss im Beispiel durch `samplepoints` dividiert werden.

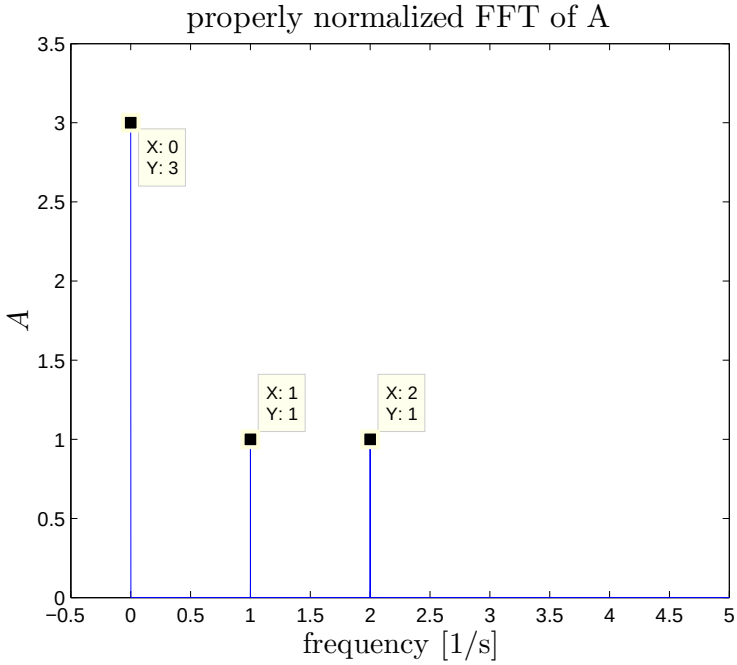

Abb. 2: korrektes Amplitudenspektrum des Zeitsignals $A(t)$.

Die Fouriertransformation erzeugt ein symmetrisches Spektrum, wobei jede im Signal enthaltene Frequenz ω im Spektrum in Form eines Peaks bei $-\omega$ und ω auftaucht. Dies ist in der Darstellung von $\cos(x)$ und $\sin(x)$ durch die komplexe Exponentialfunktion begründet:

$$\sin(x) = \frac{\mathrm{e}^{ix} - \mathrm{e}^{-ix}}{2i} \tag{2}$$

$$\cos(x) = \frac{\mathrm{e}^{ix} + \mathrm{e}^{-ix}}{2} \tag{3}$$

Die „reale Amplitude einer Frequenz ω“ verteilt sich demnach auf $-\omega$ und ω, jeweils genau zur Hälfte. Im praktischen Anwendungsfall interessiert zumeist nur eine Hälfte des Spektrums, welches dann zur korrekten Amplitudendarstellung einen Faktor 2 spendiert bekommt. Für die Frequenz 0, also für die erste Komponente des MATLAB-Vektors `freq_spectrum_A`, muss berücksichtigt werden, dass beide Spektrallinien zusammenfallen:

```
freq_spectrum_A(1,1) = freq_spectrum_A(1,1) / 2;
```

Dass `freq_spectrum_A` nun die korrekten Amplituden des Zeitsignals $A(t)$ liefert, soll durch einen Plot veranschaulicht werden. In Abb. 2 ist `freq_spectrum_A` über dem Frequenzvektor `freqs` aus Teil 2.1 aufgetragen. Die Amplituden der im Zeitsignal enthaltenen Schwingungen, wie sie im Teil 2 definiert wurden, gibt das Spektrum exakt wieder.

Die in der MATLAB-Hilfe vorgegebene (falsche) Methodik bedeutet auf unser Beispiel bezogen:

```
NFFT = 2^nextpow2(samplepoints);
nfftfreqs = linspace(0,f_sample,NFFT);
freq_spectrum_A_NFFT = 2*abs(fft(A,NFFT))/samplepoints;
freq_spectrum_A_NFFT(1,1) = freq_spectrum_A_NFFT(1,1) / 2;
```

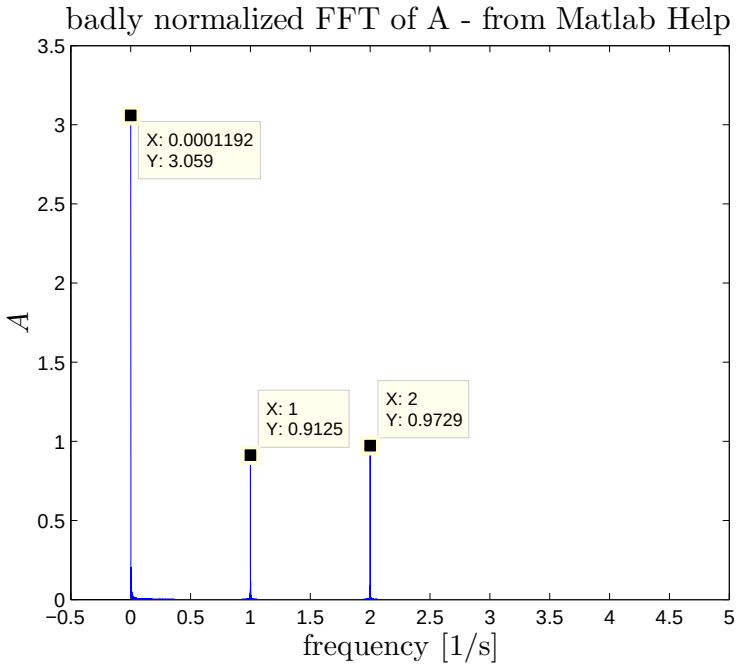

Abb. 3: fehlerbehaftetes Amplitudenspektrum des Zeitsignals $A(t)$.

In Abb. 3 ist `freq_spectrum_A_NFFT` über dem Frequenzvektor `nfftfreqs` aufgetragen. Die so bestimmten Amplituden weisen Abweichungen gegenüber den realen Amplituden auf, obwohl die Frequenzauflösung weiterhin extrem hoch gewählt ist (die selbe, die zur Bildung des exakten Spektrums führte). Diese Fehler resultieren aus der falschen Methodik, die der Autor des Beispiels in der MATLAB-Hilfe zu `fft()` anwendet, wie in Teil 2.2.2 diskutiert wurde.

3 Spektrale Untersuchung der Zeitableitung $\frac{d}{dt}A(t)$

In einigen Situationen interessiert das Spektrum der Zeitableitung des Zeitsignals $A(t)$. So z.B. im **M**agnetic **P**article **I**maging. Dort induziert die Magnetisierung ein Signal in einer Spule, sodass die Spannung als Messgröße proportional zur Zeitableitung der Magnetisierung ist. Ausgewertet wird dann die Fouriertransformierte der Spannung; im übertragenen Sinne also $FFT(\frac{d}{dt}A(t))$. In Simulationen zum MPI wird das Feld am Ort des Empfängers gebildet. Nun müsste die Ableitungsbildung erfolgen. Die numerische Implementierung einer stabilen Ableitung jedoch ist kritisch, gerade wenn, wie im Falle des MPI, die Magnetisierung analytisch gebildet wird mithilfe einer Funktion, die Singularitäten aufweist. In MATLAB gibt es die Funktion `diff()`, die aus einem N-komponentigen Vektor einen mit $N-1$ Komponenten bildet, indem sie die Differenz zwischen den Komponenten bestimmt. Damit lässt sich auf einfachste Weise eine Ableitung realisieren. In Abb. 4 ist rechts oben der zeitliche Verlauf der Magnetisierung, welche Definitionslücken bei Nulldurchgängen aufweist, dargestellt. Links unten sieht man die Ableitung, gebildet mit `diff()`.

Um die entstehenden „Ausreißer“ zu verhindern, müsste eine stabilere Ableitungsbildung realisiert werden, welche z.B. immer vier Punkte berücksichtigt und die Steigung dieser nähert. Eine performante Implementation in MATLAB (Realisierung nur durch Vektoroperationen und nicht durch Iteration über die Elemente) erscheint hier schwierig. Daher kommt es sehr gelegen, dass der Schritt der numerischen Ableitungsbildung zu sparen ist, durch einen analytischen „Trick“. Das Fourierspektrum der Größe $\frac{d}{dt}A(t)$ lässt sich durch einfache Multiplikation des Spektrums von $A(t)$ mit $i\omega$ erreichen, wie im Folgenden kurz gezeigt wird.

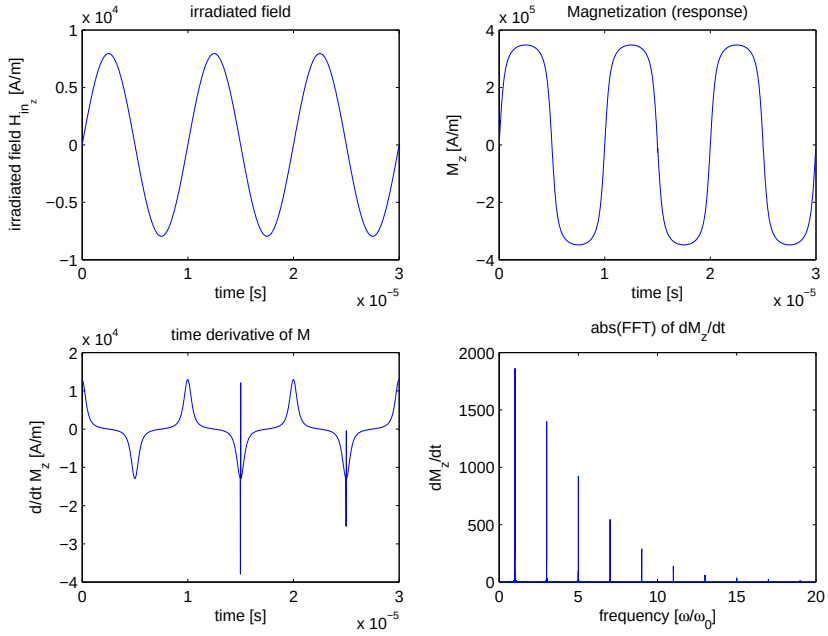

Abb. 4: rechts oben: Magnetisierungsverlauf (Definitionslücken um 0). links unten: numerische Ableitung durch einfache Differenzenbildung zweier benachbarter Punkte → „Ausreißer“ bei den Definitionslücken.

$A(t)$ sei ein periodisches Signal, sodass es durch eine Fouriersumme dargestellt werden kann:

$$A(t) = \sum_{k=0}^{\infty} a_k \cdot \mathrm{e}^{i\omega_0 kt} \tag{4}$$

Damit gilt für die Zeitableitung:

$$\frac{d}{dt}A(t) = \sum_{k=0}^{\infty} a_k \cdot \frac{d}{dt}\,\mathrm{e}^{i\omega_0 kt} \tag{5}$$

$$= i\omega_0 \sum_{k=0}^{\infty} k \cdot a_k \cdot \mathrm{e}^{i\omega_0 kt} \tag{6}$$

$$\tag{7}$$

Das Spektrum des Zeitsignals und der Zeitableitung des Zeitsignals unterscheiden sich also dadurch, dass jede Spektrallinie im Zeitableitungsspektrum einen Faktor $i\omega$ erhält, wobei ω die zur Spektrallinie zugehörige Frequenz ist.

Allgemein zeigen lässt sich dieser Zusammenhang bei Bildung der Fouriertransformierten $\mathcal{F}$ von $\frac{d}{dt}B(t)$, wobei $B(t)$ ein willkürliches Zeitsignal sei.

$$\mathcal{F}\left(\frac{d}{dt}B\right) = \int_{-\infty}^{\infty}\left(\frac{d}{dt}B\right)\mathrm{e}^{-i\omega t}\,dt \tag{8}$$

$$= \int_{-\infty}^{\infty}\left[\frac{d}{dt}B\,\mathrm{e}^{-i\omega t} + Bi\omega\,\mathrm{e}^{-i\omega t}\right]dt \tag{9}$$

$$= \frac{d}{dt}\int_{-\infty}^{\infty}B\,\mathrm{e}^{-i\omega t}\,dt + i\omega\int_{-\infty}^{\infty}B\,\mathrm{e}^{-i\omega t}\,dt \tag{10}$$

$\tilde{B}(\omega)$ sei neben $\mathcal{F}(B(t))$ ebenfalls die Fouriertransformierte von $B(t)$. Nach Gleichung 10 gilt also:

$$\mathcal{F}\left(\frac{d}{dt}B\right) = \frac{d}{dt}\tilde{B}(\omega) + i\omega\cdot\tilde{B}(\omega) = i\omega\cdot\tilde{B}(\omega) = i\omega\cdot\mathcal{F}(B) \tag{11}$$

Der Vollständigkeit halber wird jetzt das Amplitudenspektrum der numerisch bestimmten Zeitableitung von $A(t)$ mit dem Amplitudenspektrum resultierend aus obiger analytischer Betrachtung verglichen.

Zuerst wird die numerische Ableitung des Zeitsignals gebildet:

```
dt_A = diff(A) ./ Ts;
```

Diese wird fouriertransformiert und mittels der hergelittenen Regeln normiert:

```
freq_spectrum_dt_A = 2*abs(fft(dt_A))/samplepoints;
freq_spectrum_dt_A(1,1) = freq_spectrum_dt_A(1,1) / 2;
```

Das resultierende Spektrum der numerischen Ableitung ist in Abb. 5 dargestellt.

Nun wird das Fourierspektrum von $\frac{d}{dt}A(t)$ mittels Multiplikation von $\mathcal{F}(A)$ mit $i\omega$ gebildet (die imaginäre Einheit in MATLAB heißt `j`):

```
freq_spectrum_dt_A_analytic = 2 * abs(j .* 2*pi*freqs .* fft_A)/samplepoints;
freq_spectrum_dt_A_analytic(1,1) = freq_spectrum_dt_A_analytic(1,1) / 2;
```

Das resultierende Spektrum ist in Abb. 6 zu betrachten.

Beide Spektren stimmen überein und liefern das theoretisch erwartete Ergebnis. Dass die Bildung der numerischen Zeitableitung hier nicht (zumindest teilweise) versagt, liegt an gutmütigen rein harmonischen Funktionen, welche sich langsam ändern (im Vergleich zur Abtastrate) und keine Singularitäten aufweisen.

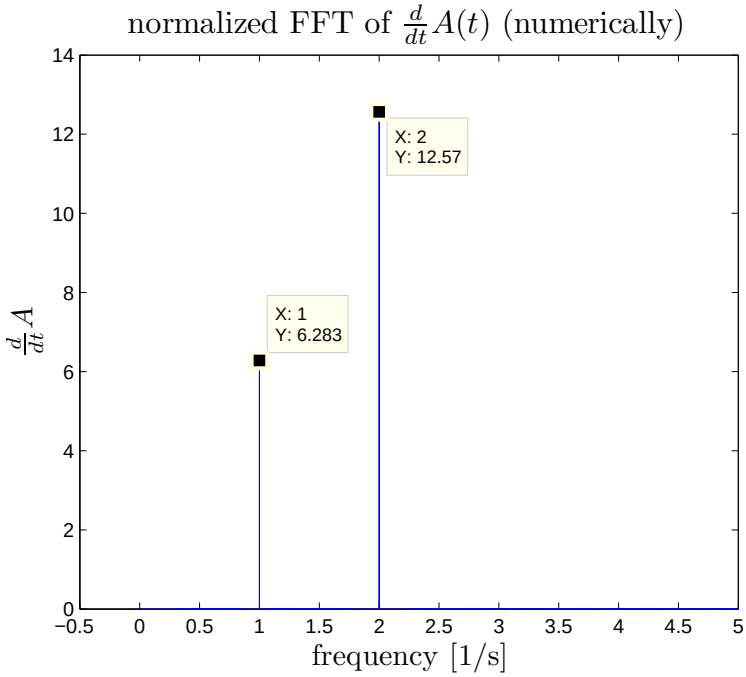

Abb. 5: Amplitudenspektrum der numerisch gewonnenen Zeitableitung von $A(t)$.

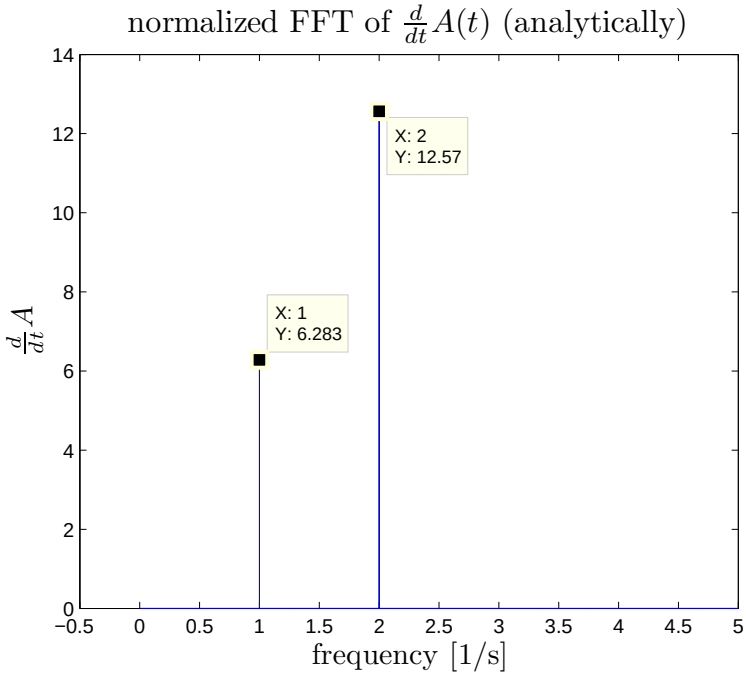

Abb. 6: Amplitudenspektrum der Zeitableitung von $A(t)$; analytisch aus dem Spektrum von $A(t)$ gewonnen durch Multiplikation mit $i\omega$.