DFT Matrix Dokumentation

Jonas Berger, Ahmed Ibrahim

17. Dezember 2022

Inhaltsverzeichnis

1 Theorie

Mithilfe der diskreten Fourier Transformation (DFT) kann der Frequenzgehalt eines zeit-diskreten Signals berechnet werden. Dadurch ist diese Transformation das Äquivalent zur Fourier Transformation (FT) für zeit-kontinuierliche Signale.

Das Kernstück der DFT ist die DFT-Matrix W, die multipliziert mit dem diskreten Zeitvektor x den Frequenzvektor X ergibt.

$$X = W \cdot x$$

Dabei wird die DFT-Matrix W aus folgenden Basisvektoren aufgebaut:

$$\mathbf{u}_0 = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}, \mathbf{u}_1 = \begin{pmatrix} 1 \\ e^{i\frac{2\pi}{N}1\cdot 1} \\ \vdots \\ e^{i\frac{2\pi}{N}1\cdot(N-1)} \end{pmatrix}, \mathbf{u}_2 = \begin{pmatrix} 1 \\ e^{i\frac{2\pi}{N}2\cdot 1} \\ \vdots \\ e^{i\frac{2\pi}{N}2\cdot(N-1)} \end{pmatrix}, \cdots, \mathbf{u}_{N-1} = \begin{pmatrix} 1 \\ e^{i\frac{2\pi}{N}(N-1)\cdot 1} \\ \vdots \\ e^{i\frac{2\pi}{N}(N-1)\cdot(N-1)} \end{pmatrix}$$

Daraus ergibt sich folgende Formel zur Berechnung der einzelnen Frequenzkomponenten X_k

$$X_k = \langle \mathbf{u}_k, \mathbf{x} \rangle = \sum_{k=0}^{N-1} e^{-i\frac{2\pi}{N}k\cdot l} x_l \quad \text{wobei } W = e^{-i\frac{2\pi}{N}k\cdot l}$$

Analog zur DFT gibt es auch die inverse DFT (IDFT) zur Berechnung der einzelnen Zeitkomponenten x_l:

$$x_l = \frac{1}{N} \sum_{k=0}^{N-1} e^{i\frac{2\pi}{N}k\cdot l} X_k \quad \text{wobei } W^{-1} = \frac{1}{N} e^{i\frac{2\pi}{N}k\cdot l}$$

2 Ermittlung einer DFT-Matrix mit unterschiedlichen Methoden

Es sollen in MATLAB Funktionen geschrieben werden, die jeweils eine DFT-Matrix mit der Größe NxN ermitteln. Implementiert wird jede Funktion unter Verwerwendung von unterschiedlichen Methoden.

2.1 Methode mit doppelter for-Schleife

Im nachfolgenden Code-Ausschnitt ist die Implementierung mit doppelter for-Schleife ersichtlich:

```
function W = dftmatrix1(N)
%Die Funktion "dftmatrix1" erzeugt eine NxN DFT-Matrix, wobei N als
%Parameter übergeben wird.
%   Input:
%     N ... Größe der Matrix
%
%   Output:
%     W ... DFT-Martrix mit der Größe NxN

W = ones(N); % Matrix mit der passenden Größe definieren
w = exp(-1i*(2*pi/N));

for k = 1:N % for-Schleife für N-Zeilen
    for l = 1:N % for-Schleife für N-Spalten
        W(k,l) = w^((k-1)*(l-1)); % Koeffizient ausrechnen
    end
end

end
```

Listing 1: Code-Implementierung mit doppelter for-Schleife

2.2 Methode mit Exponenzieren

Im nachfolgenden Code-Ausschnitt ist die Implementierung mit elementweisem Exponenzieren ersichtlich:

```
function W = dftmatrix2(N)
%Die Funktion "dftmatrix2" erzeugt eine NxN DFT-Matrix, wobei N als
%Parameter übergeben wird.
%   Input:
%     N ... Größe der Matrix
%
%   Output:
%     W ... DFT-Martrix mit der Größe NxN

An = 0:N-1;
A = An'*An; % Matrixmultiplikation des Zeilen- und Spaltenvektors ergibt die gewünschten Exponenten

W=exp((-1i*2*pi/N)*A);

end
```

Listing 2: Code-Implementierung mit Exponenzieren

2.3 Methode mit elementweisem Potenzieren

Im nachfolgenden Code-Ausschnitt ist die Implementierung mit elementweisem Potenzieren ersichtlich:

```
function W = dftmatrix3(N)
%Die Funktion "dftmatrix3" erzeugt eine NxN DFT-Matrix, wobei N als
%Parameter übergeben wird.
%  Input:
%     N ... Größe der Matrix
%
%  Output:
%     W ... DFT-Martrix mit der Größe NxN

w = exp(-1i*(2*pi/N));

An = 0:N-1;
A = An'*An; % Matrixmultiplikation des Zeilen- und Spaltenvektors ergibt die gewünschten Exponenten

W = w.^A; % elementweises Potenzieren

end
```

Listing 3: Code-Implementierung mit elementweisem Potenzieren

2.4 Verwendung der Vandermonde-Matrix

Im nachfolgenden Code-Ausschnitt ist die Implementierung mittels Vandermonde-Matrix ersichtlich:

```
function W = dftmatrix4(N)
%Die Funktion "dftmatrix4" erzeugt eine NxN DFT-Matrix, wobei N als
%Parameter übergeben wird.
%  Input:
%     N ... Größe der Matrix
%
%  Output:
%     W ... DFT-Martrix mit der Größe NxN

% Exponent initialisieren
k = 0:N-1;
k = k.*(-1i*(2*pi/N));

w = exp(k); % Exponentenvektor zur Basis von e

W = fliplr(vander(w)); % berechne DFT-Matrix/Vandermonde-Matrix

end
```

Listing 4: Code-Implementierung mittels Vandermonde-Matrix

2.5 Verwendung der Fast Fourier Transformation (FFT)

Im nachfolgenden Code-Ausschnitt ist die Implementierung mit der Fast Fourier Transformation (kurz FFT) ersichtlich:

```
function W = dftmatrix5(N)
%Die Funktion "dftmatrix5" erzeugt eine NxN DFT-Matrix, wobei N als
%Parameter übergeben wird.
%  Input:
%     N ... Größe der Matrix
%
%  Output:
%     W ... DFT-Martrix mit der Größe NxN

W = fft(eye(N)); % Fast Fourier Transformation verwenden

end
```

Listing 5: Code-Implementierung mit Fast Fourier Transformation

3 Ermittlung der Rechenzeiten der unterschiedlichen Methoden zur Erstellung einer DFT-Matrix

Es sollen nun alle, in Abschnitt 2, implementierten Funktionen miteinander im Bezug auf die Rechenzeit miteinander verglichen werden. Dazu soll ein MATLAB-Skript geschrieben werden, sodass jede individuelle Funktion eine DFT-Matrix von $N = 1$ bis $N = 2^8 = 256$ berechnet. Dabei wird mit tic und toc die Zeitdauer jeder einzelner Funktions-Ausführung gemessen. Dabei wird jede Messung 5-mal ausgeführt und der Mittelwert der ermittelten Zeit gebildet, um etwaige Schwankungen auszugleichen. Abschließend soll die Rechenzeit in Abhängikeit der Größe N der Matrix, jeder Funktion, in Form einer Grafik gegenübergestellt werden.

3.1 Code-Implementierung

```
%% init script
clear;
clc;
close all;

%%
N = 2^8; % maximal vernünftige Dauer
MAX = 5;

%% Messung 1 - dftmatrix1

elapsed_time = zeros(N,1) ;  % initilaize the elapsed times
elapsed_each = zeros(MAX,1) ;  % initilaize the elapsed times
elapsed = zeros(MAX,1) ;  % initilaize the elapsed times
disp("calculating dftmatrix1 ...");
for k = 1:N
    t1 = tic ; % starte die Stoppuhr
    for i = 1:MAX
        t2 = tic;
        W = dftmatrix1(k);
        t2 = toc(t2);
        elapsed(i) = t2;
    end
    elapsed_each(k) = mean(elapsed);
    t1 = toc(t1); % stoppe die Zeit
    elapsed_time(k) = t1;   % speichere die Zeit
end
time_avg = mean(elapsed_time)/MAX; % Berechnung der Gesamtrechenzeit (exklusiv
    Mittelwert-Berechnung)
disp("Zeit: " + time_avg + " s");
figure;
plot(1:N,elapsed_each','DisplayName','2x for'); % plot Rechenzeiten von 1x1 bis
    NxN
set(gca,'YScale','log');
xlabel('Größe der Matrix N');
ylabel('Rechenzeit in s');
axis([1 N 10^-6 10^0]);
title('Messergebnis');
legend;
grid on;
hold on;

%% Messung 2 - dftmatrix2

elapsed_time = zeros(N,1) ;  % initilaize the elapsed times
elapsed_each = zeros(MAX,1) ;  % initilaize the elapsed times
elapsed = zeros(MAX,1) ;  % initilaize the elapsed times
disp("calculating dftmatrix2 ...");
for k = 1:N
```

```
    t1 = tic ; % starte die Stoppuhr
    for i = 1:MAX
        t2 = tic;
        W = dftmatrix2(k);
        t2 = toc(t2);
        elapsed(i) = t2;
    end
    elapsed_each(k) = mean(elapsed);
    t1 = toc(t1); % stoppe die Zeit
    elapsed_time(k) = t1;   % speichere die Zeit
end
time_avg = mean(elapsed_time)/MAX; % Berechnung der Gesamtrechenzeit (exklusiv Mittelwert-Berechnung)
disp("Zeit: " + time_avg + " s");
plot(1:N,elapsed_each','DisplayName','Exponenzieren'); % plot Rechenzeiten von 1x1 bis NxN

%% Messung 3 - dftmatrix3

elapsed_time = zeros(N,1) ;  % initilaize the elapsed times
elapsed_each = zeros(MAX,1) ;  % initilaize the elapsed times
elapsed = zeros(MAX,1) ;  % initilaize the elapsed times
disp("calculating dftmatrix3 ...");
for k = 1:N
    t1 = tic ; % starte die Stoppuhr
    for i = 1:MAX
        t2 = tic;
        W = dftmatrix3(k);
        t2 = toc(t2);
        elapsed(i) = t2;
    end
    elapsed_each(k) = mean(elapsed);
    t1 = toc(t1); % stoppe die Zeit
    elapsed_time(k) = t1;   % speichere die Zeit
end
time_avg = mean(elapsed_time)/MAX; % Berechnung der Gesamtrechenzeit (exklusiv Mittelwert-Berechnung)
disp("Zeit: " + time_avg + " s");
plot(1:N,elapsed_each','DisplayName','Potenzieren'); % plot Rechenzeiten von 1x1 bis NxN

%% Messung 4 - dftmatrix4

elapsed_time = zeros(N,1) ;  % initilaize the elapsed times
elapsed_each = zeros(MAX,1) ;  % initilaize the elapsed times
elapsed = zeros(MAX,1) ;  % initilaize the elapsed times
disp("calculating dftmatrix4 ...");
for k = 1:N
    t1 = tic ; % starte die Stoppuhr
    for i = 1:MAX
        t2 = tic;
        W = dftmatrix4(k);
        t2 = toc(t2);
        elapsed(i) = t2;
    end
    elapsed_each(k) = mean(elapsed);
    t1 = toc(t1); % stoppe die Zeit
    elapsed_time(k) = t1;   % speichere die Zeit
end
time_avg = mean(elapsed_time)/MAX; % Berechnung der Gesamtrechenzeit (exklusiv Mittelwert-Berechnung)
disp("Zeit: " + time_avg + " s");
plot(1:N,elapsed_each','DisplayName','Vandermonde'); % plot Rechenzeiten von 1x1 bis NxN

%% Messung 5 - dftmatrix5

elapsed_time = zeros(N,1) ;  % initilaize the elapsed times
elapsed_each = zeros(MAX,1) ;  % initilaize the elapsed times
elapsed = zeros(MAX,1) ;  % initilaize the elapsed times
disp("calculating dftmatrix5 ...");
for k = 1:N
    t1 = tic ; % starte die Stoppuhr
```

```
    for i = 1:MAX
        t2 = tic;
        W = dftmatrix5(k);
        t2 = toc(t2);
        elapsed(i) = t2;
    end
    elapsed_each(k) = mean(elapsed);
    t1 = toc(t1); % stoppe die Zeit
    elapsed_time(k) = t1;   % speichere die Zeit
end
time_avg = mean(elapsed_time)/MAX; % Berechnung der Gesamtrechenzeit (exklusiv
    Mittelwert-Berechnung)
disp("Zeit: " + time_avg + " s");
plot(1:N,elapsed_each','DisplayName','FFT'); % plot Rechenzeiten von 1x1 bis NxN
```

Listing 6: Code-Implementierung zur Rechenzeitberechnung

3.2 Grafische Gegenüberstellung der Rechenzeiten

Die Ausführung des Codes in Abschnitt 3.1 resultiert in folgender Grafik:

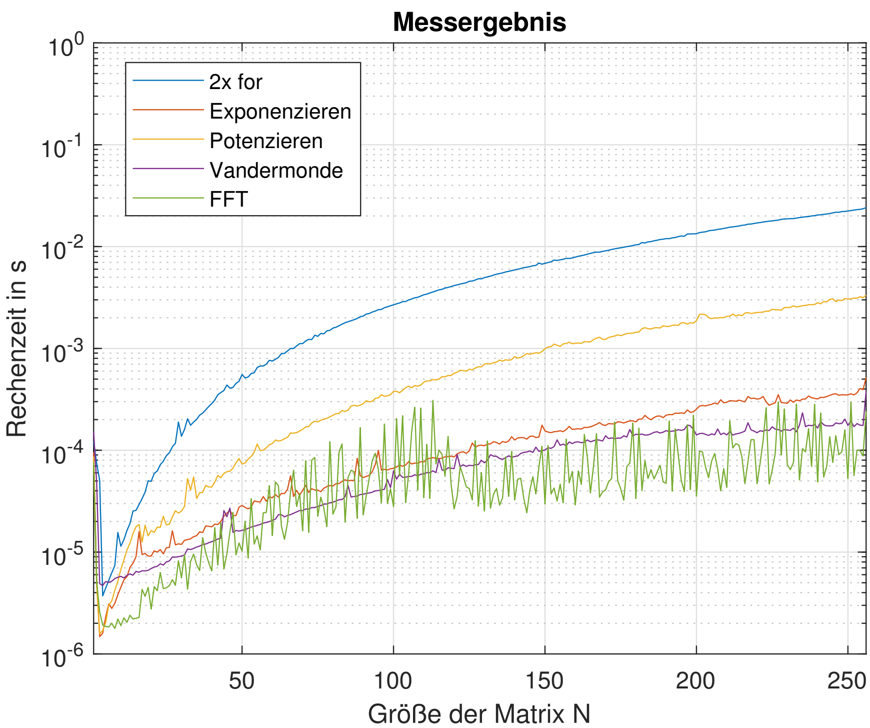

Abbildung 1: Gegenüberstellung der Rechenzeiten

Erkenntnis: Es ist deutlich zu erkennen, dass die Methode mit den doppelten for-Schleifen die langsamste Methode zur Ermittlung einer DFT-Matrix ist. Die schnellste Rechenzeit erreicht die Fast Fourier Transformation, wobei hier trotz der Mittelwertbildung eine große Schwankung der Rechenzeiten erkennbar ist. Die Methode mit der Verwendung der Vandermonde Matrix liegt zwar knapp hinter der FFT, weist aber durchgängig einen praktisch konstanten Verlauf der Rechenzeit auf.

4 Visualisierung der Basisvektoren im Einheitskreis

Die Diskrete Fourier Transformation baut auf die Basisvektoren uk auf. Diese sollen nun grafisch dargestellt werden. Hierfür werden drei Darstellungsformen gewählt: Vektoren im Einheitskreis, Punkte im Einheitskreis und Verlauf im 3-dimensionalen Raum.

4.1 Grafische Darstellungen der Basisvektoren uk

Es folgen nun die grafischen Abbildungen der Basisvektoren uk, die mittels MATLAB erzeugt werden.

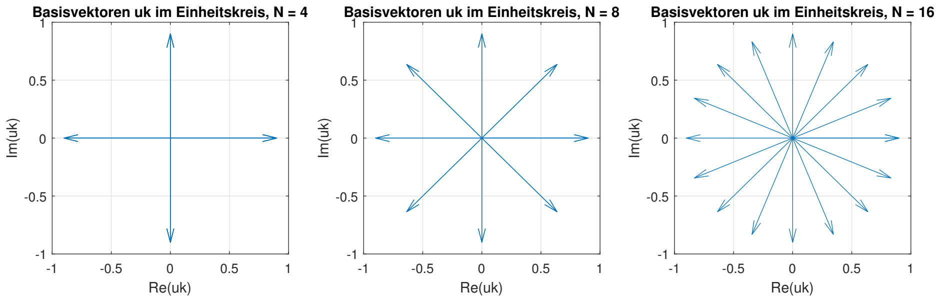

Abbildung 2: Vektoren im Einheitskreis

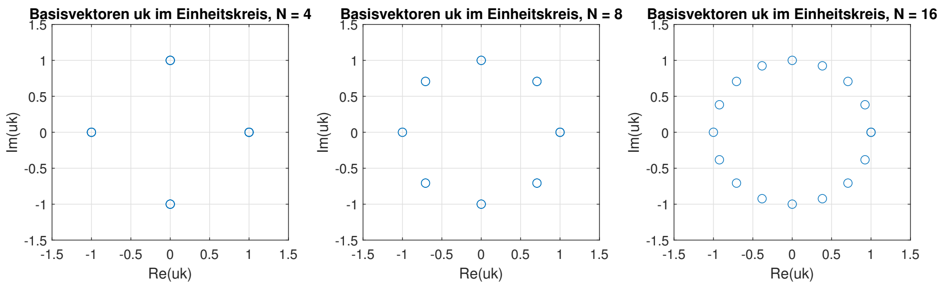

Abbildung 3: Punkte im Einheitskreis

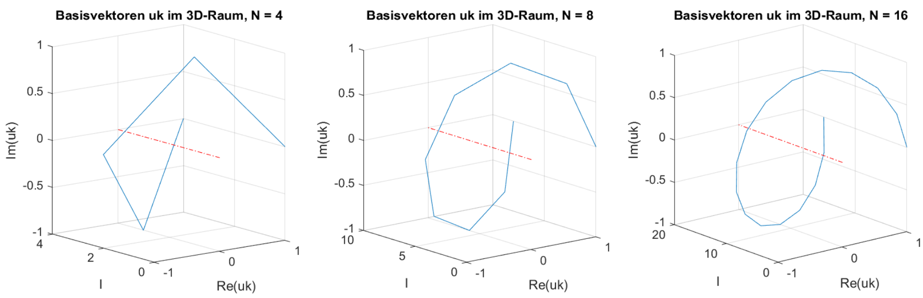

Abbildung 4: Verlauf im 3-dimensionalen Raum

Codelisting

Abbildungsverzeichnis