

# DFT Matrix Dokumentation

Jonas Berger, Ahmed Ibrahim

17. Dezember 2022

# Inhaltsverzeichnis

<b>1</b>	<b>Theorie</b>	<b>3</b>
<b>2</b>	<b>Ermittlung einer DFT-Matrix mit unterschiedlichen Methoden</b>	<b>4</b>
2.1	Methode mit doppelter for-Schleife . . . . .	4
2.2	Methode mit Exponenzieren . . . . .	4
2.3	Methode mit elementweisem Potenzieren . . . . .	5
2.4	Verwendung der Vandermonde-Matrix . . . . .	5
2.5	Verwendung der Fast Fourier Transformation (FFT) . . . . .	6
<b>3</b>	<b>Ermittlung der Rechenzeiten der unterschiedlichen Methoden zur Erstellung einer DFT-Matrix</b>	<b>7</b>
3.1	Code-Implementierung . . . . .	7
3.2	Grafische Gegenüberstellung der Rechenzeiten . . . . .	9
<b>4</b>	<b>Visualisierung der Basisvektoren im Einheitskreis</b>	<b>10</b>
4.1	Grafische Darstellungen der Basisvektoren uk . . . . .	10
	<b>Codelisting</b>	<b>11</b>
	<b>Abbildungsverzeichnis</b>	<b>11</b>

# 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:

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

Daraus ergibt sich folgende Formel zur Berechnung der einzelnen Frequenzkomponenten  $X_k$

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

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  $N \times N$  ermitteln. Implementiert wird jede Funktion unter Verwendung von unterschiedlichen Methoden.

### 2.1 Methode mit doppelter for-Schleife

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

```
1 function W = dftmatrix1(N)
2 %Die Funktion "dftmatrix1" erzeugt eine NxN DFT-Matrix, wobei N als
3 %Parameter übergeben wird.
4 % Input:
5 %   N ... Größe der Matrix
6 %
7 % Output:
8 %   W ... DFT-Matrix mit der Größe NxN
9
10 W = ones(N); % Matrix mit der passenden Größe definieren
11 w = exp(-1i*(2*pi/N));
12
13 for k = 1:N % for-Schleife für N-Zeilen
14     for l = 1:N % for-Schleife für N-Spalten
15         W(k,l) = w^((k-1)*(l-1)); % Koeffizient ausrechnen
16     end
17 end
18
19 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:

```
1 function W = dftmatrix2(N)
2 %Die Funktion "dftmatrix2" erzeugt eine NxN DFT-Matrix, wobei N als
3 %Parameter übergeben wird.
4 % Input:
5 %   N ... Größe der Matrix
6 %
7 % Output:
8 %   W ... DFT-Matrix mit der Größe NxN
9
10 An = 0:N-1;
11 A = An'*An; % Matrixmultiplikation des Zeilen- und Spaltenvektors ergibt die gewü
    nschten Exponenten
12
13 W=exp((-1i*2*pi/N)*A);
14
15 end
```

Listing 2: Code-Implementierung mit Exponenzieren

## 2.3 Methode mit elementweisem Potenzieren

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

```
1 function W = dftmatrix3(N)
2 %Die Funktion "dftmatrix3" erzeugt eine NxN DFT-Matrix, wobei N als
3 %Parameter übergeben wird.
4 % Input:
5 %   N ... Größe der Matrix
6 %
7 % Output:
8 %   W ... DFT-Matrix mit der Größe NxN
9
10 w = exp(-1i*(2*pi/N));
11
12 An = 0:N-1;
13 A = An'*An; % Matrixmultiplikation des Zeilen- und Spaltenvektors ergibt die gewü
    nschten Exponenten
14
15 W = w.^A; % elementweises Potenzieren
16
17 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:

```
1 function W = dftmatrix4(N)
2 %Die Funktion "dftmatrix4" erzeugt eine NxN DFT-Matrix, wobei N als
3 %Parameter übergeben wird.
4 % Input:
5 %   N ... Größe der Matrix
6 %
7 % Output:
8 %   W ... DFT-Matrix mit der Größe NxN
9
10 % Exponent initialisieren
11 k = 0:N-1;
12 k = k.*(-1i*(2*pi/N));
13
14 w = exp(k); % Exponentenvektor zur Basis von e
15
16 W = fliplr(vander(w)); % berechne DFT-Matrix/Vandermonde-Matrix
17
18 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:

```
1 function W = dftmatrix5(N)
2 %Die Funktion "dftmatrix5" erzeugt eine NxN DFT-Matrix, wobei N als
3 %Parameter übergeben wird.
4 % Input:
5 %   N ... Größe der Matrix
6 %
7 % Output:
8 %   W ... DFT-Martrix mit der Größe NxN
9
10 W = fft(eye(N)); % Fast Fourier Transformation verwenden
11
12 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 Abhängigkeit der Größe  $N$  der Matrix, jeder Funktion, in Form einer Grafik gegenübergestellt werden.

#### 3.1 Code-Implementierung

```

1 %% init script
2 clear;
3 clc;
4 close all;
5
6 %%
7 N = 2^8; % maximal vernünftige Dauer
8 MAX = 5;
9
10 %% Messung 1 — dftmatrix1
11
12 elapsed_time = zeros(N,1) ; % initilaize the elapsed times
13 elapsed_each = zeros(MAX,1) ; % initilaize the elapsed times
14 elapsed = zeros(MAX,1) ; % initilaize the elapsed times
15 disp("calculating dftmatrix1 ...");
16 for k = 1:N
17     t1 = tic ; % starte die Stoppuhr
18     for i = 1:MAX
19         t2 = tic;
20         W = dftmatrix1(k);
21         t2 = toc(t2);
22         elapsed(i) = t2;
23     end
24     elapsed_each(k) = mean(elapsed);
25     t1 = toc(t1); % stoppe die Zeit
26     elapsed_time(k) = t1; % speichere die Zeit
27 end
28 time_avg = mean(elapsed_time)/MAX; % Berechnung der Gesamtrechenzeit (exklusiv
    Mittelwert-Berechnung)
29 disp("Zeit: " + time_avg + " s");
30 figure;
31 plot(1:N,elapsed_each', 'DisplayName','2x for'); % plot Rechenzeiten von 1x1 bis
    NxN
32 set(gca, 'YScale', 'log');
33 xlabel('Größe der Matrix N');
34 ylabel('Rechenzeit in s');
35 axis([1 N 10^-6 10^0]);
36 title('Messergebnis');
37 legend;
38 grid on;
39 hold on;
40
41 %% Messung 2 — dftmatrix2
42
43 elapsed_time = zeros(N,1) ; % initilaize the elapsed times
44 elapsed_each = zeros(MAX,1) ; % initilaize the elapsed times
45 elapsed = zeros(MAX,1) ; % initilaize the elapsed times
46 disp("calculating dftmatrix2 ...");
47 for k = 1:N

```

```

48     t1 = tic ; % starte die Stoppuhr
49     for i = 1:MAX
50         t2 = tic;
51         W = dftmatrix2(k);
52         t2 = toc(t2);
53         elapsed(i) = t2;
54     end
55     elapsed_each(k) = mean(elapsed);
56     t1 = toc(t1); % stoppe die Zeit
57     elapsed_time(k) = t1; % speichere die Zeit
58 end
59 time_avg = mean(elapsed_time)/MAX; % Berechnung der Gesamtrechenzeit (exklusiv
    Mittelwert-Berechnung)
60 disp("Zeit: " + time_avg + " s");
61 plot(1:N,elapsed_each','DisplayName','Exponenzieren'); % plot Rechenzeiten von 1
    x1 bis NxN
62
63 %% Messung 3 — dftmatrix3
64
65 elapsed_time = zeros(N,1) ; % initilaize the elapsed times
66 elapsed_each = zeros(MAX,1) ; % initilaize the elapsed times
67 elapsed = zeros(MAX,1) ; % initilaize the elapsed times
68 disp("calculating dftmatrix3 ...");
69 for k = 1:N
70     t1 = tic ; % starte die Stoppuhr
71     for i = 1:MAX
72         t2 = tic;
73         W = dftmatrix3(k);
74         t2 = toc(t2);
75         elapsed(i) = t2;
76     end
77     elapsed_each(k) = mean(elapsed);
78     t1 = toc(t1); % stoppe die Zeit
79     elapsed_time(k) = t1; % speichere die Zeit
80 end
81 time_avg = mean(elapsed_time)/MAX; % Berechnung der Gesamtrechenzeit (exklusiv
    Mittelwert-Berechnung)
82 disp("Zeit: " + time_avg + " s");
83 plot(1:N,elapsed_each','DisplayName','Potenzieren'); % plot Rechenzeiten von 1x1
    bis NxN
84
85 %% Messung 4 — dftmatrix4
86
87 elapsed_time = zeros(N,1) ; % initilaize the elapsed times
88 elapsed_each = zeros(MAX,1) ; % initilaize the elapsed times
89 elapsed = zeros(MAX,1) ; % initilaize the elapsed times
90 disp("calculating dftmatrix4 ...");
91 for k = 1:N
92     t1 = tic ; % starte die Stoppuhr
93     for i = 1:MAX
94         t2 = tic;
95         W = dftmatrix4(k);
96         t2 = toc(t2);
97         elapsed(i) = t2;
98     end
99     elapsed_each(k) = mean(elapsed);
100    t1 = toc(t1); % stoppe die Zeit
101    elapsed_time(k) = t1; % speichere die Zeit
102 end
103 time_avg = mean(elapsed_time)/MAX; % Berechnung der Gesamtrechenzeit (exklusiv
    Mittelwert-Berechnung)
104 disp("Zeit: " + time_avg + " s");
105 plot(1:N,elapsed_each','DisplayName','Vandermonde'); % plot Rechenzeiten von 1x1
    bis NxN
106
107 %% Messung 5 — dftmatrix5
108
109 elapsed_time = zeros(N,1) ; % initilaize the elapsed times
110 elapsed_each = zeros(MAX,1) ; % initilaize the elapsed times
111 elapsed = zeros(MAX,1) ; % initilaize the elapsed times
112 disp("calculating dftmatrix5 ...");
113 for k = 1:N
114     t1 = tic ; % starte die Stoppuhr

```



```

115     for i = 1:MAX
116         t2 = tic;
117         W = dftmatrix5(k);
118         t2 = toc(t2);
119         elapsed(i) = t2;
120     end
121     elapsed_each(k) = mean(elapsed);
122     t1 = toc(t1); % stoppe die Zeit
123     elapsed_time(k) = t1; % speichere die Zeit
124 end
125 time_avg = mean(elapsed_time)/MAX; % Berechnung der Gesamtrechenzeit (exklusiv
    Mittelwert-Berechnung)
126 disp("Zeit: " + time_avg + " s");
127 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:

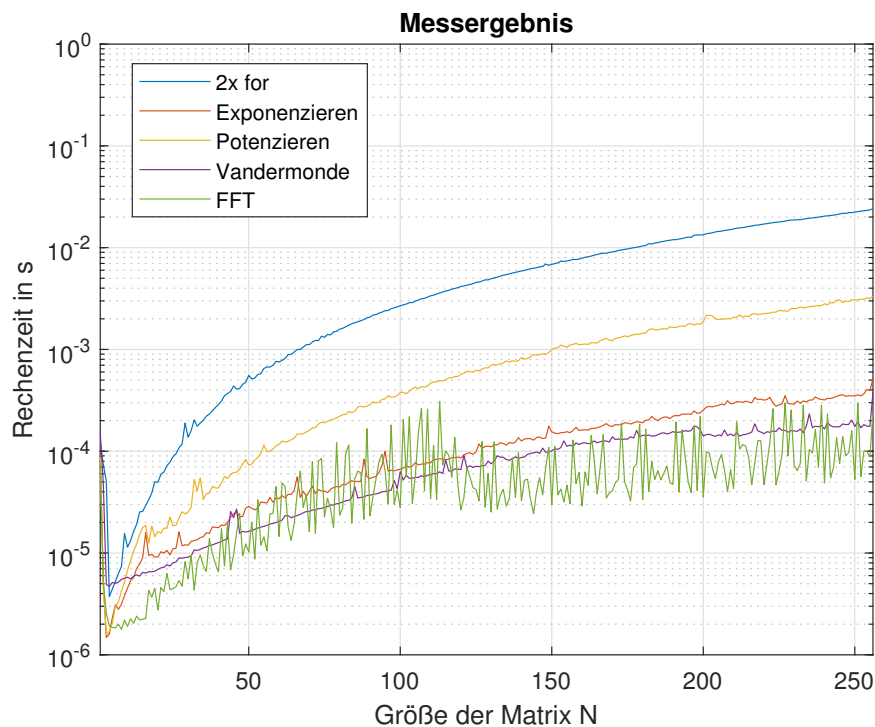


Abbildung 1: Gegenüberstellung der Rechenzeiten

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  $u_k$  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 $u_k$

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

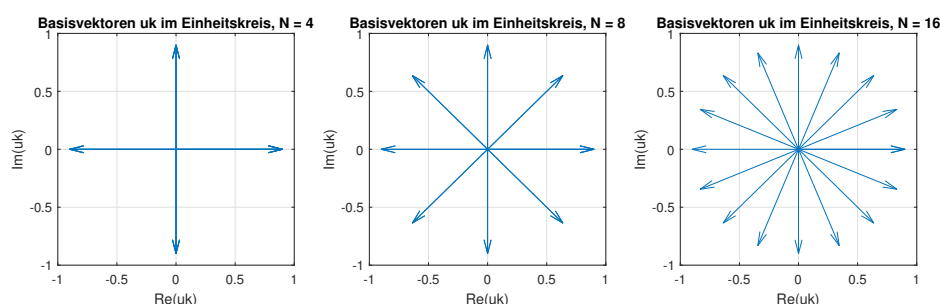


Abbildung 2: Vektoren im Einheitskreis

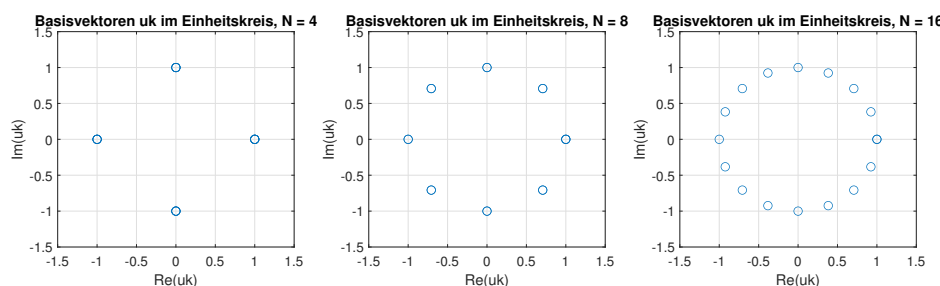


Abbildung 3: Punkte im Einheitskreis

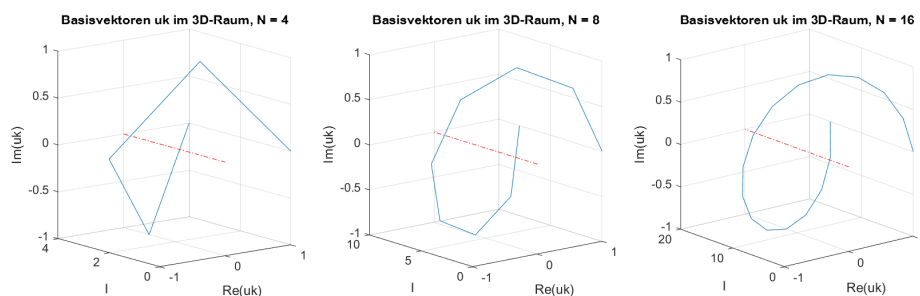


Abbildung 4: Verlauf im 3-dimensionalen Raum

## Codelisting

1	Code-Implementierung mit doppelter for-Schleife . . . . .	4
2	Code-Implementierung mit Exponenzieren . . . . .	4
3	Code-Implementierung mit elementweisem Potenzieren . . . . .	5
4	Code-Implementierung mittels Vandermonde-Matrix . . . . .	5
5	Code-Implementierung mit Fast Fourier Transformation . . . . .	6
6	Code-Implementierung zur Rechenzeitberechnung . . . . .	7

## Abbildungsverzeichnis

1	Gegenüberstellung der Rechenzeiten . . . . .	9
2	Vektoren im Einheitskreis . . . . .	10
3	Punkte im Einheitskreis . . . . .	10
4	Verlauf im 3-dimensionalen Raum . . . . .	10