14 Schnelle Fourier-Transformation

Themen:

- Polynominterpolation
- Schnelle Multiplikation

14.1 Die Lagrangesche Interpolationsaufgabe

$\mathbb{P}_n$= Raum der komplexen Polynome vom Grad $\leq n$.

14.1 Die Lagrangesche Interpolationsaufgabe

$\mathbb{P}_n$= Raum der komplexen Polynome vom Grad $\leq n$.

$x_0, \ldots, x_n$ verschiedene Stützstellen in $\mathbb{C}$

$y_0, \ldots, y_n$ nicht notwendig verschiedene Werte

14.1 Die Lagrangesche Interpolationsaufgabe

$\mathbb{P}_n$= Raum der komplexen Polynome vom Grad $\leq n$.

$x_0, \ldots, x_n$ verschiedene Stützstellen in $\mathbb{C}$

$y_0, \ldots, y_n$ nicht notwendig verschiedene Werte

In der *Lagrangeschen Interpolationsaufgabe* ist ein Polynom $p \in \mathbb{P}_n$ gesucht mit

$$p(x_j) = y_j, \quad j = 0, 1, \ldots, n.$$

14.1 Die Lagrangesche Interpolationsaufgabe

$\mathbb{P}_n$= Raum der komplexen Polynome vom Grad $\leq n$.

$x_0, \ldots, x_n$ verschiedene Stützstellen in $\mathbb{C}$

$y_0, \ldots, y_n$ nicht notwendig verschiedene Werte

In der *Lagrangeschen Interpolationsaufgabe* ist ein Polynom $p \in \mathbb{P}_n$ gesucht mit

$$p(x_j) = y_j, \quad j = 0, 1, \ldots, n.$$

Die y_j können wir uns als Werte $y_j = f(x_j)$ einer vorgegebenen Funktion f vorstellen. Wir sagen dann, dass p die Funktion f *interpoliert*.

Lagrange-Basis

Die Dimension von $\mathbb{P}_n$ ist $n+1$. Wir haben daher in der Lagrangeschen Interpolationsaufgabe $n+1$ Bedingugen gestellt, aber auch $n+1$ Freiheiten zur Verfügung.

Lagrange-Basis

Die Dimension von $\mathbb{P}_n$ ist $n+1$. Wir haben daher in der Lagrangeschen Interpolationsaufgabe $n+1$ Bedingugen gestellt, aber auch $n+1$ Freiheiten zur Verfügung.

Zur Lösung des Interpolationsproblems definieren wir die *Lagrange-Basis* $\{l_j\}_{j=0,\dots,n}$, $l_j \in \mathbb{P}_n$, durch

$$l_j(x) = \frac{(x-x_0)\dots(x-x_{j-1})(x-x_{j+1})\dots(x-x_n)}{(x_j-x_0)\dots(x_j-x_{j-1})(x_j-x_{j+1})\dots(x_j-x_n)}.$$

Die Dimension von $\mathbb{P}_n$ ist $n+1$. Wir haben daher in der Lagrangeschen Interpolationsaufgabe $n+1$ Bedingugen gestellt, aber auch $n+1$ Freiheiten zur Verfügung.

Zur Lösung des Interpolationsproblems definieren wir die *Lagrange-Basis* $\{l_j\}_{j=0,\ldots,n}$, $l_j \in \mathbb{P}_n$, durch

$$l_j(x) = \frac{(x-x_0)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_0)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)}.$$

Es gilt dann $l_i(x_j) = \delta_{ij}$

Es gilt dann $l_i(x_j) = \delta_{ij}$ und die Interpolationsaufgabe $p(x_j) = y_j$ wird gelöst durch das *Lagrangesche Interpolationspolynom*

$$p(x) = \sum_{j=0}^{n} y_j l_j(x) \in \mathbb{P}_n.$$

Es gilt dann $l_i(x_j) = \delta_{ij}$ und die Interpolationsaufgabe $p(x_j) = y_j$ wird gelöst durch das *Lagrangesche Interpolationspolynom*

$$p(x) = \sum_{j=0}^{n} y_j l_j(x) \in \mathbb{P}_n.$$

Satz Die Interpolationsaufgabe $p(x_j) = y_j$ wird eindeutig gelöst durch das Lagrangesche Interpolationspolynom.

Beweis

Gäbe es zwei Lösungen $p_1, p_2 \in \mathbb{P}_n$ von $p(x_j) = y_j$, so gilt für $q = p_1 - p_2 \in \mathbb{P}_n$, dass $q(x_j) = 0$.

Beweis

Gäbe es zwei Lösungen $p_1, p_2 \in \mathbb{P}_n$ von $p(x_j) = y_j$, so gilt für $q = p_1 - p_2 \in \mathbb{P}_n$, dass $q(x_j) = 0$.

Damit hat q $n + 1$ Nullstellen und muss das Nullpolynom sein. Daher ist $p_1 = p_2$.

Beweis

Gäbe es zwei Lösungen $p_1, p_2 \in \mathbb{P}_n$ von $p(x_j) = y_j$, so gilt für $q = p_1 - p_2 \in \mathbb{P}_n$, dass $q(x_j) = 0$.

Damit hat q $n+1$ Nullstellen und muss das Nullpolynom sein. Daher ist $p_1 = p_2$.

Beachte: Polynome vom Grad $\leq n$ werden in der Interpolation reproduziert.

14.2 Schnelle Polynommultiplikation

Sind $p, q \in \mathbb{P}_{n-1}$, so gilt mit

$$p(x) = a_{n-1}x^{n-1} + \ldots + a_1 x + a_0, \;\; q(x) = b_{n-1}x^{n-1} + \ldots + b_1 x + b_0,$$

14.2 Schnelle Polynommultiplikation

Sind $p, q \in \mathbb{P}_{n-1}$, so gilt mit

$$p(x) = a_{n-1}x^{n-1} + \ldots + a_1 x + a_0, \;\; q(x) = b_{n-1}x^{n-1} + \ldots + b_1 x + b_0,$$

für das Produkt

$$r(x) = p(x)q(x) = \sum_{l=0}^{2n-2} c_l x^l, \quad c_l = \sum_{j+k=l} a_j b_k, \;\; l = 0, \ldots, 2n-2.$$

14.2 Schnelle Polynommultiplikation

Sind $p, q \in \mathbb{P}_{n-1}$, so gilt mit

$$p(x) = a_{n-1}x^{n-1} + \ldots + a_1 x + a_0, \;\; q(x) = b_{n-1}x^{n-1} + \ldots + b_1 x + b_0,$$

für das Produkt

$$r(x) = p(x)q(x) = \sum_{l=0}^{2n-2} c_l x^l, \quad c_l = \sum_{j+k=l} a_j b_k, \;\; l = 0, \ldots, 2n-2.$$

Es wurde die Konvention $a_j, b_j = 0$ für $j > n-1$ verwendet.

14.2 Schnelle Polynommultiplikation

$$r(x) = p(x)q(x) = \sum_{l=0}^{2n-2} c_l x^l, \quad c_l = \sum_{j+k=l} a_j b_k, \; l = 0, \ldots, 2n-2.$$

$$r(x) = p(x)q(x) = \sum_{l=0}^{2n-2} c_l x^l, \quad c_l = \sum_{j+k=l} a_j b_k, \; l = 0, \ldots, 2n-2.$$

Da jeder Koeffizient des einen Polynoms mit jedem Koeffizienten des anderen Polynoms multipliziert wird, benötigen wir für die direkte Polynommultiplikation:

n^2 Multiplikationen und $O(n^2)$ Additionen.

Alternative Interpolation

Eine Alternative zur direkten Multiplikation ist die Polynominterpolation nach Lagrange.

Alternative Interpolation

Eine Alternative zur direkten Multiplikation ist die Polynominterpolation nach Lagrange.

Demnach ist ein Polynom vom Grade $n - 1$ durch die Werte $p(x_j)$ an n verschiedenen Stützstellen $x_1, \ldots, x_n$ eindeutig bestimmt.

Alternative Interpolation

Eine Alternative zur direkten Multiplikation ist die Polynominterpolation nach Lagrange.

Demnach ist ein Polynom vom Grade $n-1$ durch die Werte $p(x_j)$ an n verschiedenen Stützstellen $x_1, \dots, x_n$ eindeutig bestimmt.

Algorithmus:

- Werte die Polynome p und q an $2n-1$ verschiedenen Stellen aus.

Alternative Interpolation

Eine Alternative zur direkten Multiplikation ist die Polynominterpolation nach Lagrange.

Demnach ist ein Polynom vom Grade $n-1$ durch die Werte $p(x_j)$ an n verschiedenen Stützstellen $x_1, \ldots, x_n$ eindeutig bestimmt.

Algorithmus:

- Werte die Polynome p und q an $2n-1$ verschiedenen Stellen aus.
- Bestimme die $2n-1$ Produkte $p(x_j)q(x_j)$.

Alternative Interpolation

Eine Alternative zur direkten Multiplikation ist die Polynominterpolation nach Lagrange.

Demnach ist ein Polynom vom Grade $n-1$ durch die Werte $p(x_j)$ an n verschiedenen Stützstellen $x_1, \ldots, x_n$ eindeutig bestimmt.

Algorithmus:

- Werte die Polynome p und q an $2n-1$ verschiedenen Stellen aus.
- Bestimme die $2n-1$ Produkte $p(x_j)q(x_j)$.
- Bestimme das Interpolationspolynom zu diesen Produkten.

Alternative Interpolation

Da das Interpolationspolynom eindeutig bestimmt ist, wird durch diese Vorgehensweise das Produktpolynom reproduziert.

Alternative Interpolation

Da das Interpolationspolynom eindeutig bestimmt ist, wird durch diese Vorgehensweise das Produktpolynom reproduziert.

Dieses Verfahren ist zunächst weniger effektiv als die direkte Methode:

- Für die Auswertung von $p(x_j)$ werden $O(n)$ Operationen benötigt.

Da das Interpolationspolynom eindeutig bestimmt ist, wird durch diese Vorgehensweise das Produktpolynom reproduziert.

Dieses Verfahren ist zunächst weniger effektiv als die direkte Methode:

- Für die Auswertung von $p(x_j)$ werden $O(n)$ Operationen benötigt.
- Benötigen diese Auswertungen für $O(n)$ Punkte.

Alternative Interpolation

Da das Interpolationspolynom eindeutig bestimmt ist, wird durch diese Vorgehensweise das Produktpolynom reproduziert.

Dieses Verfahren ist zunächst weniger effektiv als die direkte Methode:

- Für die Auswertung von $p(x_j)$ werden $O(n)$ Operationen benötigt.
- Benötigen diese Auswertungen für $O(n)$ Punkte.
- Das Aufstellen des Interpolationspolynoms zu $p(x_j)q(x_j)$ kostet ebenfalls $O(n^2)$ Operationen.

Komplexe Einheitswurzeln

Um ein Polynom an n Stellen auszuwerten, verwenden wir die komplexen Einheitswurzeln

$$\omega_n = \exp\left(\frac{2\pi i}{n}\right), \quad \omega_n^k = \exp\left(\frac{2k\pi i}{n}\right), \; k = 0, \ldots, n-1.$$

Komplexe Einheitswurzeln

Um ein Polynom an n Stellen auszuwerten, verwenden wir die komplexen Einheitswurzeln

$$\omega_n = \exp\left(\frac{2\pi i}{n}\right), \quad \omega_n^k = \exp\left(\frac{2k\pi i}{n}\right), \; k = 0, \dots, n-1.$$

Für diese gilt

$$\omega_n^k = \omega_n^{k+ln} \quad \forall l \in \mathbb{Z}$$

wegen $\exp(z) = \exp(z + 2\pi i)$.

Die grundlegende Idee

Sei nun n gerade und p ein Polynom vom Grade $\leq n-1$.

Die grundlegende Idee

Sei nun n gerade und p ein Polynom vom Grade $\leq n-1$. Wir schreiben

$$\begin{aligned}
p(x) &= a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2x^2 + a_1x + a_0 \\
&= (a_{n-2}x^{n-2} + \ldots + a_2x^2 + a_0) + x(a_{n-1}x^{n-2} + \ldots + a_3x^2 + a_1) \\
&= p_g(x^2) + xp_u(x^2).
\end{aligned}$$

Die grundlegende Idee

Sei nun n gerade und p ein Polynom vom Grade $\leq n-1$. Wir schreiben

$$\begin{aligned} p(x) &= a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2x^2 + a_1x + a_0 \\ &= (a_{n-2}x^{n-2} + \ldots + a_2x^2 + a_0) + x(a_{n-1}x^{n-2} + \ldots + a_3x^2 + a_1) \\ &= p_g(x^2) + xp_u(x^2). \end{aligned}$$

Um $p(\omega_n^k)$, $k = 0, \ldots, n-1$ zu bestimmen, müssen die Polynome p_g und p_u an den Stellen $\omega_n^{2k} = \omega_{n/2}^k$ ausgewertet werden.

Die grundlegende Idee

Sei nun n gerade und p ein Polynom vom Grade $\leq n-1$. Wir schreiben

$$\begin{aligned} p(x) &= a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2x^2 + a_1x + a_0 \\ &= (a_{n-2}x^{n-2} + \ldots + a_2x^2 + a_0) + x(a_{n-1}x^{n-2} + \ldots + a_3x^2 + a_1) \\ &= p_g(x^2) + xp_u(x^2). \end{aligned}$$

Um $p(\omega_n^k)$, $k = 0, \ldots, n-1$ zu bestimmen, müssen die Polynome p_g und p_u an den Stellen $\omega_n^{2k} = \omega_{n/2}^k$ ausgewertet werden.

Dies sind

$$\omega_{n/2}^0, \ldots, \omega_{n/2}^{n/2-1}.$$

Die grundlegende Idee

$$p(x) = p_g(x^2) + x p_u(x^2).$$

Die grundlegende Idee

$$p(x) = p_g(x^2) + x p_u(x^2).$$

Werte p_g und p_u an den Stellen $\omega_n^{2k} = \omega_{n/2}^k$ aus.

Die grundlegende Idee

$$p(x) = p_g(x^2) + xp_u(x^2).$$

Werte p_g und p_u an den Stellen $\omega_n^{2k} = \omega_{n/2}^k$ aus.

p_g und p_u besitzen nur noch den Grad $n/2 - 1$!

Die grundlegende Idee

$$p(x) = p_g(x^2) + xp_u(x^2).$$

Werte p_g und p_u an den Stellen $\omega_n^{2k} = \omega_{n/2}^k$ aus.

p_g und p_u besitzen nur noch den Grad $n/2 - 1$!

Ferner lassen sich die Auswertungen von p_g und p_u zweimal verwenden, nämlich für $k = 0, \ldots, n/2 - 1$ und für $k + n/2$ wegen

$$\omega_n^{2k} = \omega_n^{2k+2n/2}.$$

Auswertung des Polynoms = FFT

Da das beschriebene Verfahren offenbar rekursiv durchgeführt werden kann, wenn n eine Zweierpotenz ist, nehmen wir nun $n = 2^l$ an.

Da das beschriebene Verfahren offenbar rekursiv durchgeführt werden kann, wenn n eine Zweierpotenz ist, nehmen wir nun $n = 2^l$ an.

Das Programm

$$\textbf{recursive subroutine } FFT(n, p, \omega, a)$$

bestimmt die Auswertung eines Polynoms vom Grade $n-1$ in den n Punkten $\omega_n^0, \ldots, \omega_n^{n-1}$.

Da das beschriebene Verfahren offenbar rekursiv durchgeführt werden kann, wenn n eine Zweierpotenz ist, nehmen wir nun $n = 2^l$ an.

Das Programm

$$\textbf{recursive subroutine } FFT(n, p, \omega, a)$$

bestimmt die Auswertung eines Polynoms vom Grade $n-1$ in den n Punkten $\omega_n^0, \ldots, \omega_n^{n-1}$.

Input: $n = 2^l$ ist die Länge des Problems, p ist das auszuwertende Polynom als n-Vektor $p = (a_0, \ldots, a_{n-1})$ gespeichert und $\omega = \omega_n$..

Auswertung des Polynoms = FFT

Da das beschriebene Verfahren offenbar rekursiv durchgeführt werden kann, wenn n eine Zweierpotenz ist, nehmen wir nun $n = 2^l$ an.

Das Programm

$$\textbf{recursive subroutine } FFT(n, p, \omega, a)$$

bestimmt die Auswertung eines Polynoms vom Grade $n-1$ in den n Punkten $\omega_n^0, \ldots, \omega_n^{n-1}$.

Input: $n = 2^l$ ist die Länge des Problems, p ist das auszuwertende Polynom als n-Vektor $p = (a_0, \ldots, a_{n-1})$ gespeichert und $\omega = \omega_n$..

Output: der n-Vektor $a = (p(\omega^0), \ldots, p(\omega^{n-1}))$.

```
recursive subroutine FFT(n, p, ω, a)
  if n = 1 then
    a(0) = p(0)
  else
    n2 = n/2
    pg = (a0, a2, ..., a(n-2))
    pu = (a1, a3, ..., a(n-1))
    call FFT(n2, pg, ω^2, g)
    call FFT(n2, pu, ω^2, u)
    do k = 0, n2 - 1
      a(k) = g(k) + ω^k u(k)
      a(k + n2) = g(k) - ω^k u(k)   !*
    enddo
  endif
end
```

FFT

recursive subroutine $FFT(n, p, \omega, a)$
  if $n = 1$ **then**
    $a(0) = p(0)$
  else
    $n_2 = n/2$
    $p_g = (a_0, a_2, \ldots, a_{n-2})$
    $p_u = (a_1, a_3, \ldots, a_{n-1})$
    call $FFT(n_2, p_g, \omega^2, g)$
    call $FFT(n_2, p_u, \omega^2, u)$
    do $k = 0, n_2 - 1$
      $a(k) = g(k) + \omega^k u(k)$
      $a(k + n_2) = g(k) - \omega^k u(k)$ $\quad !*$
    enddo
  endif
end

$*$: $\omega_n^{k+n/2} = \omega_n^k \omega_n^{n/2} = -\omega_n^k$

Multiplikationen bei FFT

Bestimmen die Anzahl der Multiplikationen $M(n)$ für diesen Algorithmus.

Multiplikationen bei FFT

Bestimmen die Anzahl der Multiplikationen $M(n)$ für diesen Algorithmus.

ω^k können einmal berechnet und anschließend abgespeichert werden.

Bestimmen die Anzahl der Multiplikationen $M(n)$ für diesen Algorithmus.

ω^k können einmal berechnet und anschließend abgespeichert werden.

Multiplikationen fallen nur in

$$\begin{aligned}
&\textbf{do } k = 0, n_2 - 1 \\
&\quad a(k) = g(k) + \omega^k u(k) \\
&\quad a(k + n_2) = g(k) - \omega^k u(k) \\
&\textbf{enddo}
\end{aligned}$$

an.

Multiplikationen bei FFT

Bestimmen die Anzahl der Multiplikationen $M(n)$ für diesen Algorithmus.

ω^k können einmal berechnet und anschließend abgespeichert werden.

Multiplikationen fallen nur in

$$
\begin{array}{l}
\textbf{do } k = 0, n_2 - 1 \\
\quad a(k) = g(k) + \omega^k u(k) \\
\quad a(k + n_2) = g(k) - \omega^k u(k) \\
\textbf{enddo}
\end{array}
$$

an.

Daher: Nur die $n/2$ Multiplikationen $\omega^k u(k)$!

Multiplikationen bei FFT

Da *FFT* zweimal mit Inputlänge $n/2$ aufgerufen wird, genügt $M(n)$ der Rekursion

$$M(n) = 2M(\frac{n}{2}) + \frac{n}{2}$$

Multiplikationen bei FFT

Da *FFT* zweimal mit Inputlänge $n/2$ aufgerufen wird, genügt $M(n)$ der Rekursion

$$M(n) = 2M(\frac{n}{2}) + \frac{n}{2}$$

für $n = 2^l$ daher

$$M(2^l) = 2M(2^{l-1}) + 2^{l-1}.$$

Multiplikationen bei FFT

Da *FFT* zweimal mit Inputlänge $n/2$ aufgerufen wird, genügt $M(n)$ der Rekursion

$$M(n) = 2M(\frac{n}{2}) + \frac{n}{2}$$

für $n = 2^l$ daher

$$M(2^l) = 2M(2^{l-1}) + 2^{l-1}.$$

Iterieren wir diese Beziehung l mal, so erhalten wir

$$M(2^l) = l2^{l-1} + M(1)2^l.$$

Multiplikationen bei FFT

$$M(2^l) = l2^{l-1} + M(1)2^l.$$

Multiplikationen bei FFT

$$M(2^l) = l2^{l-1} + M(1)2^l.$$

Es ist $M(1) = 0$ und daher

$$M(n) = \frac{n}{2} \log n,$$

wobei mit log der Logarithmus zur Basis 2 bezeichnet wird.

Multiplikationen bei FFT

$$M(2^l) = l2^{l-1} + M(1)2^l.$$

Es ist $M(1) = 0$ und daher

$$M(n) = \frac{n}{2} \log n,$$

wobei mit log der Logarithmus zur Basis 2 bezeichnet wird.

Anzahl der Addditionen ist analog. Wir erhalten für die Gesamtzahl an Operationen ebenfalls $O(n \log n)$.

Nun wenden wir uns dem Interpolationsproblem zu:

Zu $b_0, \ldots, b_{n-1}$ suche ein Polynom $p \in \mathbb{P}_{n-1}$ mit

$$p(\omega_n^k) = b_k \text{ für } k = 0, \ldots, n-1.$$

Nun wenden wir uns dem Interpolationsproblem zu:

$$\text{Zu } b_0, \ldots, b_{n-1} \text{ suche ein Polynom } p \in \mathbb{P}_{n-1} \text{ mit}$$

$$p(\omega_n^k) = b_k \text{ für } k = 0, \ldots, n-1.$$

Wie bereits im vorigen Abschnitt gezeigt wurde, existiert ein solches Polynom und ist eindeutig bestimmt.

Nun wenden wir uns dem Interpolationsproblem zu:

$$\text{Zu } b_0, \ldots, b_{n-1} \text{ suche ein Polynom } p \in \mathbb{P}_{n-1} \text{ mit}$$

$$p(\omega_n^k) = b_k \text{ für } k = 0, \ldots, n-1.$$

Wie bereits im vorigen Abschnitt gezeigt wurde, existiert ein solches Polynom und ist eindeutig bestimmt.

Um dieses Problem anzugehen, deuten wir zunächst die Polynomauswertung als Multiplikation einer Matrix mit einem Vektor.

Das Interpolationsproblem = FFI

Zu $\alpha = (\alpha_0, \dots, \alpha_{n-1})^T \in \mathbb{C}^n$ mit paarweise verschiedenen α_k definieren wir die zugehörige *Vandermondsche Matrix* als

$$V(\alpha) = \begin{pmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^{n-1} \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{n-1} & \alpha_{n-1}^2 & \cdots & \alpha_{n-1}^{n-1} \end{pmatrix}.$$

Das Interpolationsproblem = FFI

$$V(\alpha) = \begin{pmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^{n-1} \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{n-1} & \alpha_{n-1}^2 & \cdots & \alpha_{n-1}^{n-1} \end{pmatrix}.$$

Für ein Polynom

$$p(x) = \sum_{j=0}^{n-1} a_j x^j$$

Das Interpolationsproblem = FFI

$$V(\alpha) = \begin{pmatrix} 1 & \alpha_0 & \alpha_0^2 & \cdots & \alpha_0^{n-1} \\ 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{n-1} & \alpha_{n-1}^2 & \cdots & \alpha_{n-1}^{n-1} \end{pmatrix}.$$

Für ein Polynom

$$p(x) = \sum_{j=0}^{n-1} a_j x^j$$

gilt dann

$$b := \big(p(\alpha_0), \ldots, p(\alpha_{n-1})\big)^T = V(\alpha)a, \quad a = (a_0, \ldots, a_{n-1})^T.$$

Das Interpolationsproblem = FFI

$$b := \big(p(\alpha_0), \ldots, p(\alpha_{n-1})\big)^T = V(\alpha)a, \quad a = (a_0, \ldots, a_{n-1})^T.$$

Die Auswertung von p an den Stellen $\alpha_0, \ldots, \alpha_{n-1}$ ist also nichts anderes als die Bestimmung von $V(\alpha)a$.

Das Interpolationsproblem = FFI

$$b := \big(p(\alpha_0), \ldots, p(\alpha_{n-1})\big)^T = V(\alpha)a, \quad a = (a_0, \ldots, a_{n-1})^T.$$

Die Auswertung von p an den Stellen $\alpha_0, \ldots, \alpha_{n-1}$ ist also nichts anderes als die Bestimmung von $V(\alpha)a$.

Die Rekonstruktion der Koeffizienten $a_0, \ldots, a_{n-1}$ aus den Daten $b_k = p(\alpha_k)$ (=Interpolation) ist das dazu inverse Problem und wird durch

$$a = V(\alpha)^{-1} b$$

gelöst.

Bei der speziellen Wahl $\alpha = (\omega_n^0, \ldots, \omega_n^{n-1})^T$ kann die inverse Matrix zu $V(\alpha)$ leicht angegeben werden.

Bei der speziellen Wahl $\alpha = (\omega_n^0, \ldots, \omega_n^{n-1})^T$ kann die inverse Matrix zu $V(\alpha)$ leicht angegeben werden.

Für eine komplexe Zahl β schreiben wir

$$[\beta] = (\beta^0, \ldots, \beta^{n-1})^T.$$

Das Interpolationsproblem = FFI

Bei der speziellen Wahl $\alpha = (\omega_n^0, \ldots, \omega_n^{n-1})^T$ kann die inverse Matrix zu $V(\alpha)$ leicht angegeben werden.

Für eine komplexe Zahl β schreiben wir

$$[\beta] = (\beta^0, \ldots, \beta^{n-1})^T.$$

Satz Für die n-te Einheitswurzel $\omega_n = \exp(2\pi i/n)$ gilt

$$V([\omega_n])^{-1} = \frac{1}{n} V([\omega_n^{-1}]).$$

Beweis

Für die Matrix $W = V([\omega_n])V([\omega_n^{-1}])$ gilt

$$w_{jk} = \sum_{l=0}^{n-1} \omega_n^{jl}\omega_n^{-kl} = \sum_{l=0}^{n-1} (\omega_n^{j-k})^l.$$

Beweis

Für die Matrix $W = V([\omega_n])V([\omega_n^{-1}])$ gilt

$$w_{jk} = \sum_{l=0}^{n-1} \omega_n^{jl} \omega_n^{-kl} = \sum_{l=0}^{n-1} (\omega_n^{j-k})^l.$$

Für $j = k$ erhalten wir $w_{jj} = n$. Für $j \neq k$ ist $0 < |j - k| < n$ und damit $\omega_n^{j-k} \neq 1$.

Beweis

Für die Matrix $W = V([\omega_n])V([\omega_n^{-1}])$ gilt

$$w_{jk} = \sum_{l=0}^{n-1} \omega_n^{jl}\omega_n^{-kl} = \sum_{l=0}^{n-1} (\omega_n^{j-k})^l.$$

Für $j = k$ erhalten wir $w_{jj} = n$. Für $j \neq k$ ist $0 < |j-k| < n$ und damit $\omega_n^{j-k} \neq 1$.

Wir können daher die geometrische Summenformel anwenden, also

$$\sum_{l=0}^{n-1} (\omega_n^{j-k})^l = \frac{\omega_n^{n(j-k)} - 1}{\omega_n^{j-k} - 1} = 0$$

wegen $\omega_n^{n(j-k)} = (\omega_n^n)^{j-k} = 1$.

Der Algorithmus zur Bestimmung des Interpolationspolynoms $p \in \mathbb{P}_{n-1}$ aus den Daten $b := \big(p(1), p(\omega_n^1), \ldots, p(\omega_n^{n-1})\big)$:

$$\textbf{subroutine } \mathit{FFI}(n, b, \omega, p)$$

Das Interpolationsproblem = FFI

Der Algorithmus zur Bestimmung des Interpolationspolynoms $p \in \mathbb{P}_{n-1}$ aus den Daten $b := \big(p(1), p(\omega_n^1), \ldots, p(\omega_n^{n-1})\big)$:

$$\textbf{subroutine}\ FFI(n, b, \omega, p)$$

Input: die Länge $n = 2^l$, der n-Vektor b und $\omega = \omega_n$.

Der Algorithmus zur Bestimmung des Interpolationspolynoms $p \in \mathbb{P}_{n-1}$ aus den Daten $b := \big(p(1), p(\omega_n^1), \ldots, p(\omega_n^{n-1})\big)$:

$$\textbf{subroutine } FFI(n, b, \omega, p)$$

Input: die Länge $n = 2^l$, der n-Vektor b und $\omega = \omega_n$.

Output: Das gesuchte Polynom auf dem n-Vektor $p = (a_0, \ldots, a_{n-1})$.

Der Algorithmus zur Bestimmung des Interpolationspolynoms $p \in \mathbb{P}_{n-1}$ aus den Daten $b := \big(p(1), p(\omega_n^1), \ldots, p(\omega_n^{n-1})\big)$:

subroutine $FFI(n, b, \omega, p)$

Input: die Länge $n = 2^l$, der n-Vektor b und $\omega = \omega_n$.

Output: Das gesuchte Polynom auf dem n-Vektor $p = (a_0, \ldots, a_{n-1})$.

subroutine $FFI(n, b, \omega, p)$
  call $FFT(n, b, \omega^{-1}, p)$
  $p = n^{-1} p$
end

Schnelle Multiplikation von Polynomen

Seien $p, q \in \mathbb{P}_k$ Ziel: $r = p \cdot q$

Schnelle Multiplikation von Polynomen

Seien $p, q \in \mathbb{P}_k$ Ziel: $r = p \cdot q$

Bestimme die kleinste Zahl l mit $n = 2^l > 2k$ und rufen mit $\omega = \omega_n$

$$FFT(n, p, \omega, a), \quad FFT(n, q, \omega, b)$$

auf.

Seien $p, q \in \mathbb{P}_k$ Ziel: $r = p \cdot q$

Bestimme die kleinste Zahl l mit $n = 2^l > 2k$ und rufen mit $\omega = \omega_n$

$$FFT(n, p, \omega, a), \quad FFT(n, q, \omega, b)$$

auf.

Berechne $c(k) = a(k)b(k)$ für $k = 0, \ldots, n-1$, was n Multiplikationen entspricht.

Seien $p, q \in \mathbb{P}_k$ Ziel: $r = p \cdot q$

Bestimme die kleinste Zahl l mit $n = 2^l > 2k$ und rufen mit $\omega = \omega_n$

$$FFT(n, p, \omega, a), \quad FFT(n, q, \omega, b)$$

auf.

Berechne $c(k) = a(k)b(k)$ für $k = 0, \dots, n-1$, was n Multiplikationen entspricht.

Mit dem Aufruf von

$$FFI(n, c, \omega, r)$$

stehen auf dem Vektor r die Koeffizienten des gesuchten Produkts $p(x)q(x)$.

Seien $p, q \in \mathbb{P}_k$ Ziel: $r = p \cdot q$

Bestimme die kleinste Zahl l mit $n = 2^l > 2k$ und rufen mit $\omega = \omega_n$

$$FFT(n, p, \omega, a), \quad FFT(n, q, \omega, b)$$

auf.

Berechne $c(k) = a(k)b(k)$ für $k = 0, \ldots, n-1$, was n Multiplikationen entspricht.

Mit dem Aufruf von

$$FFI(n, c, \omega, r)$$

stehen auf dem Vektor r die Koeffizienten des gesuchten Produkts $p(x)q(x)$.

Der gesamte Algorithmus benötigt immer noch $O(n \log n)$ und wegen $n \leq 4k$ auch $O(k \log k)$ Rechenoperationen.

14.3 Schnelle Multiplikation natürlicher Zahlen

Liegen natürliche Zahlen a, b in einem Stellenwertsystem zur Basis g vor und besitzen diese Zahlen eine Länge $\leq K$, so lassen sie sich mit $O(K^2)$ Operationen multiplizieren.

14.3 Schnelle Multiplikation natürlicher Zahlen

Liegen natürliche Zahlen a, b in einem Stellenwertsystem zur Basis g vor und besitzen diese Zahlen eine Länge $\leq K$, so lassen sie sich mit $O(K^2)$ Operationen multiplizieren.

Eine Operation besteht aus der Multiplikation zweier Ziffern oder der Addition zweier ganzer Zahlen.

14.3 Schnelle Multiplikation natürlicher Zahlen

Liegen natürliche Zahlen a, b in einem Stellenwertsystem zur Basis g vor und besitzen diese Zahlen eine Länge $\leq K$, so lassen sie sich mit $O(K^2)$ Operationen multiplizieren.

Eine Operation besteht aus der Multiplikation zweier Ziffern oder der Addition zweier ganzer Zahlen.

Alternativ können wir a und b die Polynome

$$p_a(x) = \sum_{i=0}^{K-1} a_i x^i, \quad p_b(x) = \sum_{i=0}^{K-1} b_i x^i$$

zuordnen, wobei $a_i, b_i \in \{0, 1 \ldots, g-1\}$ die zugehörigen Ziffern sind.

14.3 Schnelle Multiplikation natürlicher Zahlen

$$p_a(x) = \sum_{i=0}^{K-1} a_i x^i, \quad p_b(x) = \sum_{i=0}^{K-1} b_i x^i$$

mit „Ziffern“ $a_i, b_i \in \{0, 1 \ldots, g-1\}$.

14.3 Schnelle Multiplikation natürlicher Zahlen

$$p_a(x) = \sum_{i=0}^{K-1} a_i x^i, \quad p_b(x) = \sum_{i=0}^{K-1} b_i x^i$$

mit „Ziffern“ $a_i, b_i \in \{0, 1 \ldots, g-1\}$.

Mit der schnellen Polynommultiplikation kann

$$p_c(x) = p_a(x) p_b(x)$$

bestimmt werden.

14.3 Schnelle Multiplikation natürlicher Zahlen

$$p_a(x) = \sum_{i=0}^{K-1} a_i x^i, \quad p_b(x) = \sum_{i=0}^{K-1} b_i x^i$$

mit „Ziffern“ $a_i, b_i \in \{0, 1 \ldots, g-1\}$.

Mit der schnellen Polynommultiplikation kann

$$p_c(x) = p_a(x) p_b(x)$$

bestimmt werden.

Entferne aus $p_c(g)$ den Übertrag - fertig!

Vorsicht: Rundungsfehler

Mit dem im vorigen Abschnitt dargestellten Multiplikationsverfahren erhalten wir aufgrund von Rundungsfehlern ein Polynom $\tilde{p}_c$, dessen Koeffizienten Gleitkommazahlen sind.

Vorsicht: Rundungsfehler

Mit dem im vorigen Abschnitt dargestellten Multiplikationsverfahren erhalten wir aufgrund von Rundungsfehlern ein Polynom $\tilde{p}_c$, dessen Koeffizienten Gleitkommazahlen sind.

Die verwendete Gleitkommaarithmetik muss so ausgelegt sein, dass aus $\tilde{p}_c$ durch Rundung das korrekte Polynom $p_c \in \mathbb{N}$ entsteht.

Wir betrachten den Spezialfall

$$K = 2^k \quad \text{und} \quad g = 2^l,$$

mit dem Zahlen der Bitlänge $n \leq \frac{1}{2}Kl$ miteinander multipliziert werden können.

Anforderung an die Arithmetik

Wir betrachten den Spezialfall

$$K = 2^k \quad \text{und} \quad g = 2^l,$$

mit dem Zahlen der Bitlänge $n \leq \frac{1}{2}Kl$ miteinander multipliziert werden können.

Verwenden wir für die schnelle Fouriertransformation eine Gleitkommaarithmetik der Genauigkeit 2^{-m}, so erhält man nach Rundung von $\tilde{p}_c$ das exakte Polynom p_c, falls

$$m \geq 3k + 2l + \log k + 7/2.$$

Anforderung an die Arithmetik

$$m \geq 3k + 2l + \log k + 7/2.$$

Um Zahlen der Bitlänge

$$n = 2^{13} = 8192$$

miteinander zu multiplizieren, können wir hier $l = 8$, $k = 11$ setzen und erhalten $m = 55$.

Anforderung an die Arithmetik

$$m \geq 3k + 2l + \log k + 7/2.$$

Um Zahlen der Bitlänge

$$n = 2^{13} = 8192$$

miteinander zu multiplizieren, können wir hier $l = 8$, $k = 11$ setzen und erhalten $m = 55$.

Das wird von einer doppelt genauen Gleitkommaarithmetik geleistet.

Anforderung an die Arithmetik

$$m \geq 3k + 2l + \log k + 7/2.$$

Um Zahlen der Bitlänge

$$n = 2^{13} = 8192$$

miteinander zu multiplizieren, können wir hier $l = 8$, $k = 11$ setzen und erhalten $m = 55$.

Das wird von einer doppelt genauen Gleitkommaarithmetik geleistet.

Für die meisten praktisch relevanten Fälle lassen sich damit zwei n-stellige Zahlen in $O(n \log n)$ Gleitkommaoperationen miteinander multiplizieren.

Anforderung an die Arithmetik

Für die Multiplikation noch größerer Zahlen kann auch die Gleitkommamultiplikation mit Hilfe der schnellen Fouriertransformation beschleunigt werden.

Anforderung an die Arithmetik

Für die Multiplikation noch größerer Zahlen kann auch die Gleitkommamultiplikation mit Hilfe der schnellen Fouriertransformation beschleunigt werden.

Für den Gesamtalgorithmus haben wir dann eine Komplexität von

$$O(n \log n \log(\log n))$$

die allerdings auch nur bis zu einer sehr großen Zahl n richtig ist.

In der *modalen Fouriertransformation* von Schönhage und Strassen (1971) werden kommutative Ringe mit n-ter Einheitswurzel verwendet, die letztlich aus ganzen Zahlen bestehen.

Modalen Fouriertransformation

In der *modalen Fouriertransformation* von Schönhage und Strassen (1971) werden kommutative Ringe mit n-ter Einheitswurzel verwendet, die letztlich aus ganzen Zahlen bestehen.

Damit kann man zwei n-stellige Zahlen in $O(n \log n \log(\log n))$ miteinander multiplizieren, in diesem Fall ohne Einschränkung an n.

14.4 Diskrete Fourier-Transformation und Bildverarbeitung

Im folgenden schreiben wir einen Vektor des $\mathbb{C}^n$ in der Form

$$f = (f(0), f(1), \ldots, f(n-1)).$$

Im folgenden schreiben wir einen Vektor des $\mathbb{C}^n$ in der Form

$$f = (f(0), f(1), \ldots, f(n-1)).$$

Wir stellen uns f als eine eindimensionale stückweise konstante Funktion vor oder noch besser als die (reellen) Grauwerte eines eindimensionalen Bildes.

Im folgenden schreiben wir einen Vektor des $\mathbb{C}^n$ in der Form

$$f = (f(0), f(1), \ldots, f(n-1)).$$

Wir stellen uns f als eine eindimensionale stückweise konstante Funktion vor oder noch besser als die (reellen) Grauwerte eines eindimensionalen Bildes.

n wird immer als geradzahlig vorausgesetzt.

Diskrete Wellenfunktionen

Für $k = 0, \ldots, n-1$ definieren wir die diskreten Wellenfunktionen

$$\phi_k(j) = \exp\left(\frac{2\pi ik}{n} j\right), \quad j = 0, \ldots, n-1, \quad i = \sqrt{-1}.$$

Diskrete Wellenfunktionen

Für $k = 0, \ldots, n-1$ definieren wir die diskreten Wellenfunktionen

$$\phi_k(j) = \exp\left(\frac{2\pi i k}{n} j\right), \quad j = 0, \ldots, n-1, \quad i = \sqrt{-1}.$$

Die Größe

$$\theta_k = \frac{2\pi k}{n}$$

ist die *Frequenz* der Wellenfunktion ϕ_k.

Diskrete Wellenfunktionen

Für $k = 0, \dots, n-1$ definieren wir die diskreten Wellenfunktionen

$$\phi_k(j) = \exp\left(\frac{2\pi i k}{n} j\right), \quad j = 0, \dots, n-1, \quad i = \sqrt{-1}.$$

Die Größe

$$\theta_k = \frac{2\pi k}{n}$$

ist die *Frequenz* der Wellenfunktion ϕ_k.

Die Funktion ϕ_0 ist konstant 1 und $\phi_1(j)$ führt für $j = 0, \dots, n-1$ eine volle Cosinus- bzw. Sinus-Schwingung in Real- bzw. Imaginärteil durch.

Diskrete Wellenfunktionen

Für beliebiges $k \geq 1$ gilt

$$\begin{aligned}\phi_{n-k}(j) &= \exp(2\pi ij(n-k)/n) \\ &= \exp(2\pi ij)\exp(-2\pi ijk/n) = \overline{\phi_k(j)}.\end{aligned}$$

Für beliebiges $k \geq 1$ gilt

$$\begin{aligned}\phi_{n-k}(j) &= \exp(2\pi ij(n-k)/n) \\ &= \exp(2\pi ij)\exp(-2\pi ijk/n) = \overline{\phi_k(j)}.\end{aligned}$$

Für $k < n/2$ spannen daher ϕ_k und ϕ_{n-k} den zweidimensionalen reellen Raum

$$\mathrm{span}\,\{\cos(2\pi jk/n), \sin(2\pi jk/n)\}$$

auf.

Für beliebiges $k \geq 1$ gilt

$$\begin{aligned}\phi_{n-k}(j) &= \exp(2\pi ij(n-k)/n) \\ &= \exp(2\pi ij)\exp(-2\pi ijk/n) = \overline{\phi_k(j)}.\end{aligned}$$

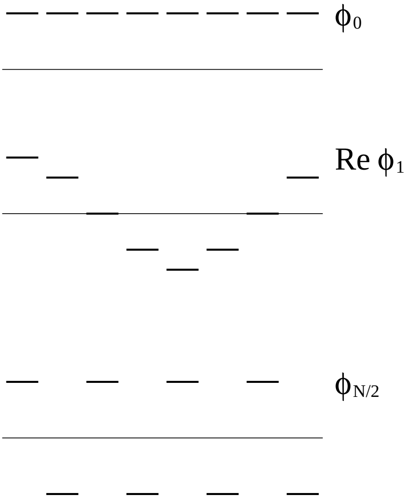
Für $k < n/2$ spannen daher ϕ_k und ϕ_{n-k} den zweidimensionalen reellen Raum

$$\mathrm{span}\,\{\cos(2\pi jk/n), \sin(2\pi jk/n)\}$$

auf.

Für kleine k entsprechen ϕ_k, ϕ_{n-k} langen Wellen, zu $k \sim n/2$ gehören kurze Wellen.

Diskrete Wellenfunktionen



Orthogonalität der Wellenfunktionen

Auf dem $\mathbb{C}^n$ verwenden wir das Standardprodukt

$$(f, g)_n = \sum_{j=0}^{n-1} f(j)\,\overline{g(j)}.$$

Orthogonalität der Wellenfunktionen

Auf dem $\mathbb{C}^n$ verwenden wir das Standardprodukt

$$(f, g)_n = \sum_{j=0}^{n-1} f(j)\,\overline{g(j)}.$$

Satz Die Funktionen ϕ_k, $k = 0, \dots, n-1$, bilden eine Orthogonalbasis des $\mathbb{C}^n$, es gilt

$$(\phi_k, \phi_l)_n = n\delta_{kl}.$$

Beweis

Wir zeigen zunächst

$$\sum_{k=0}^{n-1} \exp(2\pi i k r/n) = \begin{cases} n & \text{falls } r = ln \text{ für ein } l \in \mathbb{Z} \\ 0 & \text{für alle sonstigen ganzzahligen } r \end{cases} .$$

Beweis

Wir zeigen zunächst

$$\sum_{k=0}^{n-1} \exp(2\pi ikr/n) = \begin{cases} n & \text{falls } r = ln \text{ für ein } l \in \mathbb{Z} \\ 0 & \text{für alle sonstigen ganzzahligen } r \end{cases} .$$

Für $r = ln$ ist dies klar.

Beweis

Wir zeigen zunächst

$$\sum_{k=0}^{n-1} \exp(2\pi ikr/n) = \begin{cases} n & \text{falls } r = ln \text{ für ein } l \in \mathbb{Z} \\ 0 & \text{für alle sonstigen ganzzahligen } r \end{cases} .$$

Für $r = ln$ ist dies klar.

Für $r \neq ln$ können wir mit

$$\exp(2\pi ikr/n) = \exp(2\pi ir/n)^k = \omega^k, \quad \omega = \omega^1 \neq 1,$$

wie zuvor die geometrische Summenformel anwenden,

$$\sum_{k=0}^{n-1} \exp(2\pi ikr/n) = \sum_{k=0}^{n-1} \omega^k = \frac{\omega^n - 1}{\omega - 1} = 0.$$

Beweis

$$\sum_{k=0}^{n-1} \exp(2\pi i k r/n) = \begin{cases} n & \text{falls } r = ln \text{ für ein } l \in \mathbb{Z} \\ 0 & \text{für alle sonstigen ganzzahligen } r \end{cases} .$$

Beweis

$$\sum_{k=0}^{n-1} \exp(2\pi i k r/n) = \begin{cases} n & \text{falls } r = ln \text{ für ein } l \in \mathbb{Z} \\ 0 & \text{für alle sonstigen ganzzahligen } r \end{cases} .$$

Daraus folgt für $k, l = 0, \ldots, n-1$

$$(\phi_k, \phi_l)_n = \sum_{j=0}^{n-1} \exp(2\pi i j k/n) \exp(-2\pi i j l/n)$$

$$= \sum_{j=0}^{n-1} \exp(2\pi i j(k-l)/n) = n\delta_{kl}.$$

Ist $\{v_k\}$ eine Orthogonalbasis eines komplexen endlich dimensionalen Raumes mit Produkt $(\cdot, \cdot)$, so gilt für jeden Vektor u dieses Raumes

$$u = \sum_k \alpha_k v_k .$$

Fourierkoeffizienten

Ist $\{v_k\}$ eine Orthogonalbasis eines komplexen endlich dimensionalen Raumes mit Produkt $(\cdot,\cdot)$, so gilt für jeden Vektor u dieses Raumes

$$u = \sum_k \alpha_k v_k .$$

Wir multiplizieren diese Identität skalar mit v_l und erhalten wegen der Orthogonalität der v_k

$$(u, v_l) = \sum_k (\alpha_k v_k, v_l) = \alpha_l (v_l, v_l)$$

Fourierkoeffizienten

Ist $\{v_k\}$ eine Orthogonalbasis eines komplexen endlich dimensionalen Raumes mit Produkt $(\cdot, \cdot)$, so gilt für jeden Vektor u dieses Raumes

$$u = \sum_k \alpha_k v_k.$$

Wir multiplizieren diese Identität skalar mit v_l und erhalten wegen der Orthogonalität der v_k

$$(u, v_l) = \sum_k (\alpha_k v_k, v_l) = \alpha_l (v_l, v_l)$$

und daher

$$\alpha_l = \frac{(u, v_l)}{(v_l, v_l)}.$$

Im Spezialfall der diskreten Fourier-Transformation sind die v_k die Wellenfunktionen ϕ_k.

Im Spezialfall der diskreten Fourier-Transformation sind die v_k die Wellenfunktionen ϕ_k.

Die *Fourier-Transformierte* $\mathcal{F}_n f \in \mathbb{C}^n$ ist

$$\mathcal{F}_n f(l) = \frac{1}{n}(f, \phi_l)_n = \frac{1}{n}\sum_{j=0}^{n-1} f(j) \exp(-2\pi i j l/n), \quad l = 0, \ldots, n-1.$$

Im Spezialfall der diskreten Fourier-Transformation sind die v_k die Wellenfunktionen ϕ_k.

Die *Fourier-Transformierte* $\mathcal{F}_n f \in \mathbb{C}^n$ ist

$$\mathcal{F}_n f(l) = \frac{1}{n}(f, \phi_l)_n = \frac{1}{n}\sum_{j=0}^{n-1} f(j)\exp(-2\pi ijl/n), \quad l = 0, \ldots, n-1.$$

$\mathcal{F}_n$ ordnet jeder diskreten Funktion f die Amplituden der Wellenfunktionen zu: $\mathcal{F}_n f(l)$ ist der Koeffizient α_l in der Darstellung

$$f = \sum_l \alpha_l \phi_l.$$

Umkehrung der Diskreten Fourier-Transformation

Es gilt für alle j

$$(\mathcal{F}_n f, \overline{\phi_j}) = \frac{1}{n}\sum_{l=0}^{n-1}\left((f,\phi_l),\overline{\phi_j}\right) = \frac{1}{n}\sum_{l,r=0}^{n-1} f(r)\overline{\phi_l(r)}\phi_j(l)$$

$$= \frac{1}{n}\sum_{l,r=0}^{n-1} f(r)\overline{\phi_r(l)}\phi_j(l)$$

Es gilt für alle j

$$(\mathcal{F}_n f, \overline{\phi_j}) = \frac{1}{n} \sum_{l=0}^{n-1} ((f, \phi_l), \overline{\phi_j}) = \frac{1}{n} \sum_{l,r=0}^{n-1} f(r) \overline{\phi_l(r)} \phi_j(l)$$

$$= \frac{1}{n} \sum_{l,r=0}^{n-1} f(r) \overline{\phi_r(l)} \phi_j(l) = \frac{1}{n} \sum_{r=0}^{n-1} f(r) (\phi_j, \phi_r)$$

$$= f(j)$$

Bemerkung

- Manchmal wird der Faktor $\frac{1}{n}$ durch $1/\sqrt{n}$ ersetzt und in der Rücktransformation ein Faktor $1/\sqrt{n}$ hinzugefügt.

Bemerkung

- Manchmal wird der Faktor $\frac{1}{n}$ durch $1/\sqrt{n}$ ersetzt und in der Rücktransformation ein Faktor $1/\sqrt{n}$ hinzugefügt.
- Dann stehen sich der „Originalraum“ und der „Fourier-Raum“ vollständig symmetrisch gegenüber.

Wir ordnen f das Polynom

$$p_f(x) = f(n-1)x^{n-1} + \ldots + f(1)x + f(0)$$

zu.

Wir ordnen f das Polynom

$$p_f(x) = f(n-1)x^{n-1} + \ldots + f(1)x + f(0)$$

zu.

Dann gilt wegen $\exp(jr) = \exp(r)^j$

$$\mathcal{F}_n f(l) = \frac{1}{n} p_f(\exp(-2\pi i l/n)).$$

Wir ordnen f das Polynom

$$p_f(x) = f(n-1)x^{n-1} + \ldots + f(1)x + f(0)$$

zu.

Dann gilt wegen $\exp(jr) = \exp(r)^j$

$$\mathcal{F}_n f(l) = \frac{1}{n} p_f(\exp(-2\pi i l/n)).$$

Mit $\omega_n = \exp(2\pi i/n)$ ist damit p_f an den Stellen $\overline{\omega}_n^l$ auszuwerten.

Wir ordnen f das Polynom

$$p_f(x) = f(n-1)x^{n-1} + \ldots + f(1)x + f(0)$$

zu.

Dann gilt wegen $\exp(jr) = \exp(r)^j$

$$\mathcal{F}_n f(l) = \frac{1}{n} p_f(\exp(-2\pi i l/n)).$$

Mit $\omega_n = \exp(2\pi i/n)$ ist damit p_f an den Stellen $\overline{\omega}_n^l$ auszuwerten.

Dies entspricht genau dem Algrithmus FFI aus dem letzten Abschnitt.

Berechnung der Rücktransformation

Die Rücktransformation ist analog durch Auswertung eines Polynoms an den Stellen ω_n^l gegeben, was der Algorithmus FFT leistet.

Berechnung der Rücktransformation

Die Rücktransformation ist analog durch Auswertung eines Polynoms an den Stellen ω_n^l gegeben, was der Algorithmus FFT leistet.

Wir können daher Fourier-Transformation und Rücktransformation in $O(n \log n)$ Operationen durchführen.

14.5 Abklingverhalten und diskrete Cosinus-Transformation

Ist ein Datensatz

$$f(0), \dots, f(n-1)$$

strukturlos, kann nicht komprimiert werden.

14.5 Abklingverhalten und diskrete Cosinus-Transformation

Ist ein Datensatz

$$f(0), \ldots, f(n-1)$$

strukturlos, kann nicht komprimiert werden.

Stellt aber f beispielsweise die Grautöne eines eindimensionalen Bildes dar, so wird f über weite Strecken nur wenig variieren mit Ausnahme von einigen Kanten.

14.5 Abklingverhalten und diskrete Cosinus-Transformation

Ist ein Datensatz

$$f(0), \ldots, f(n-1)$$

strukturlos, kann nicht komprimiert werden.

Stellt aber f beispielsweise die Grautöne eines eindimensionalen Bildes dar, so wird f über weite Strecken nur wenig variieren mit Ausnahme von einigen Kanten.

Wir bezeichnen eine solche Funktion als glatt oder stückweise glatt.

14.5 Abklingverhalten und diskrete Cosinus-Transformation

Ist ein Datensatz

$$f(0), \ldots, f(n-1)$$

strukturlos, kann nicht komprimiert werden.

Stellt aber f beispielsweise die Grautöne eines eindimensionalen Bildes dar, so wird f über weite Strecken nur wenig variieren mit Ausnahme von einigen Kanten.

Wir bezeichnen eine solche Funktion als glatt oder stückweise glatt.

Bei glatten Funktionen klingen die Amplituden, die zu Wellenfunktionen mit hoher Frequenz gehören, schneller ab als bei nichtglatten Funktionen.

14.5 Abklingverhalten und diskrete Cosinus-Transformation

Als erste Idee für eine Bildkompression liegt es nahe, eine Fourier-Transformation durchzuführen, jedoch nur die langwelligen Amplituden abzuspeichern.

14.5 Abklingverhalten und diskrete Cosinus-Transformation

Als erste Idee für eine Bildkompression liegt es nahe, eine Fourier-Transformation durchzuführen, jedoch nur die langwelligen Amplituden abzuspeichern.

Bei glatten Bildern ist eine solche Kompression mit bloßem Auge nicht mehr sichtbar, sie wird allerdings an sehr scharfen Kanten entlarvt, die durch die Kompression etwas verschmiert werden.

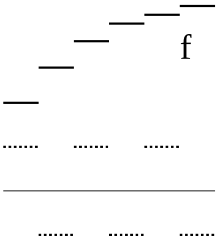
14.5 Abklingverhalten und diskrete Cosinus-Transformation

Als erste Idee für eine Bildkompression liegt es nahe, eine Fourier-Transformation durchzuführen, jedoch nur die langwelligen Amplituden abzuspeichern.

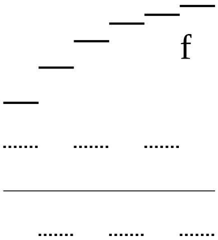
Bei glatten Bildern ist eine solche Kompression mit bloßem Auge nicht mehr sichtbar, sie wird allerdings an sehr scharfen Kanten entlarvt, die durch die Kompression etwas verschmiert werden.

Aus diesem Grund werden in der JPEG-Kompression alle Amplituden berücksichtigt, die höheren Frequenzen aber nur näherungsweise abgespeichert.

Beispiel

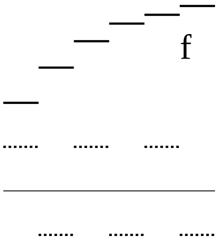


Beispiel



In der obigen Zeichnung sehen wir eine glatte Funktion f und $\phi_{n/2}$, das ist die am schnellsten oszillierende Wellenfunktion.

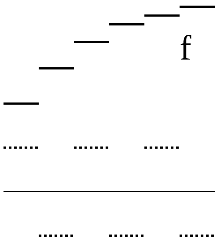
Beispiel



In der obigen Zeichnung sehen wir eine glatte Funktion f und $\phi_{n/2}$, das ist die am schnellsten oszillierende Wellenfunktion.

Aufgrund des glatten Verhaltens von f heben sich die Summanden in $\sum f(j)\phi_{n/2}(j)$ fast gegenseitig auf.

Beispiel

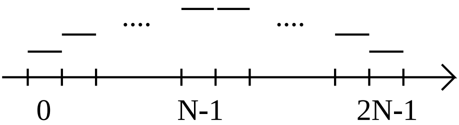


Aus dem Bild geht aber auch hervor, dass die zugehörige Amplitude noch kleiner wäre, wenn auch die auf $\mathbb{Z}$ periodisch fortgesetzte Funktion

$$f(j + k(n-1)) = f(j), \quad k \in \mathbb{Z},$$

glatt ist.

Symmetrische Spiegelung



Um die Glattheitseigenschaften der fortgesetzten Funktion zu verbessern, wird f durch symmetrische Spiegelung zu einer Funktion in $\mathbb{C}^{2n}$ fortgesetzt:

$$f(2n-1-j) = f(j), \quad j = 0, \ldots, n-1.$$

Fourier-Transformation der fortgesetzten Funktion

- Verwende Fourier-Transformation auf $\mathbb{C}^{2N}$ für dieses fortgesetzte f.

Fourier-Transformation der fortgesetzten Funktion

- Verwende Fourier-Transformation auf $\mathbb{C}^{2N}$ für dieses fortgesetzte f.
- Es muss möglich sein, mit n Fourier-Koeffizienten für die exakte Darstellung von f auszukommen.

Fourier-Transformation der fortgesetzten Funktion

- Verwende Fourier-Transformation auf $\mathbb{C}^{2N}$ für dieses fortgesetzte f.
- Es muss möglich sein, mit n Fourier-Koeffizienten für die exakte Darstellung von f auszukommen.
- Ist f reellwertig, so muss auch eine reellwertige Darstellung der Fourier-Koeffizienten erreicht werden können.

Wir verwenden daher die *Cosinus-Transformation*

$$\mathcal{C}_n f(l) = c(l) \sum_{j=0}^{n-1} f(j) \cos\left(\frac{\pi(2j+1)l}{2n}\right)$$

Wir verwenden daher die *Cosinus-Transformation*

$$\mathcal{C}_n f(l) = c(l) \sum_{j=0}^{n-1} f(j) \cos\left(\frac{\pi(2j+1)l}{2n}\right)$$

mit Rücktransformation

$$f(j) = \sum_{l=0}^{n-1} c(l)\mathcal{C}_n f(l) \cos\left(\frac{\pi(2j+1)l}{2n}\right),$$

mit $c(0) = 1/\sqrt{n}$ und $c(l) = \sqrt{2/n}$ für $l \neq 0$.

Denn die reellen Wellenfunktionen

$$\psi_l(j) = \cos\left(\frac{\pi(2j+1)l}{2n}\right), \quad l = 0, \ldots, n-1$$

bilden ein Orthogonalsystem bezüglich des Produkts

$$(f, g) = \sum_{j=0}^{n-1} f(j)g(j).$$

Denn die reellen Wellenfunktionen

$$\psi_l(j) = \cos\left(\frac{\pi(2j+1)l}{2n}\right), \quad l = 0, \ldots, n-1$$

bilden ein Orthogonalsystem bezüglich des Produkts

$$(f, g) = \sum_{j=0}^{n-1} f(j)g(j).$$

Klar wegen

$$\psi_l = \frac{1}{2}(\phi_l + \overline{\phi_l}).$$

Berechnung der Cosinus-Transformation

Die Cosinus-Transformation wird aus der Fourier-Transformierten für die symmetrisch fortgesetzte Funktion f bestimmt.

Die Cosinus-Transformation wird aus der Fourier-Transformierten für die symmetrisch fortgesetzte Funktion f bestimmt.

In

$$2n\mathcal{F}_{2n}f(l) = \sum_{j=0}^{2n-1} f(j)\exp\left(\frac{-2\pi ijl}{2n}\right)$$

multiplizieren wir beide Seiten mit $\exp(-\pi il/(2n))$.

Die Cosinus-Transformation wird aus der Fourier-Transformierten für die symmetrisch fortgesetzte Funktion f bestimmt.

In

$$2n\mathcal{F}_{2n}f(l) = \sum_{j=0}^{2n-1} f(j)\exp\left(\frac{-2\pi ijl}{2n}\right)$$

multiplizieren wir beide Seiten mit $\exp(-\pi il/(2n))$.

Erhalte für die Koeffizienten wegen $f(2n-1-j) = f(j)$:

Berechnung der Cosinus-Transformation

$$\exp\left(\frac{-\pi i l}{2n}\right)\left(\exp\left(\frac{-2\pi i j l}{2n}\right)+\exp\left(\frac{-2\pi i(2n-1-j)l}{2n}\right)\right)$$

Berechnung der Cosinus-Transformation

$$
\begin{aligned}
\exp\big(\frac{-\pi i l}{2n}\big)&\Big(\exp\big(\frac{-2\pi i j l}{2n}\big) + \exp\big(\frac{-2\pi i(2n-1-j)l}{2n}\big)\Big) \\
&= \exp\big(\frac{-\pi i l}{2n}\big)\Big(\exp\big(\frac{-2\pi i j l}{2n}\big) + \exp\big(\frac{2\pi i(j+1)l}{2n}\big)\Big) \\
&= \exp\big(\frac{-\pi i(2j+1)l}{2n}\big) + \exp\big(\frac{\pi i(2j+1)l}{2n}\big) \\
&= 2\cos\big(\frac{\pi(2j+1)l}{2n}\big).
\end{aligned}
$$

Algorithmus

Wir erhalten damit folgenden Algorithmus für die schnelle Cosinus-Transformation:

Algorithmus

Wir erhalten damit folgenden Algorithmus für die schnelle Cosinus-Transformation:

Seien $f(0), \ldots, f(n-1) \in \mathbb{R}$ gegeben.

Wir erhalten damit folgenden Algorithmus für die schnelle Cosinus-Transformation:

Seien $f(0), \ldots, f(n-1) \in \mathbb{R}$ gegeben.

1. Erweitere den Datensatz durch symmetrische Spiegelung zu

$$\big(f(0), f(1), \ldots, f(n-1), f(n-1), \ldots, f(1), f(0)\big) \in \mathbb{R}^{2n}.$$

Algorithmus

Wir erhalten damit folgenden Algorithmus für die schnelle Cosinus-Transformation:

Seien $f(0), \dots, f(n-1) \in \mathbb{R}$ gegeben.

1. Erweitere den Datensatz durch symmetrische Spiegelung zu

$$\big(f(0), f(1), \dots, f(n-1), f(n-1), \dots, f(1), f(0)\big) \in \mathbb{R}^{2n}.$$

2. Wende schnelle Fourier-Interpolation FFI auf den erweiterten Datensatz an. Erhalte

$$\mathcal{F}_{2n} f(l), \quad l = 0, \dots, 2n-1.$$

3. Bestimme die Koeffizienten der Cosinus-Transformation aus

$$\mathcal{C}_n f(l) = \frac{c(l)n}{2} \exp\Big(\frac{-\pi i l}{2n}\Big) \mathcal{F}_{2n} f(l) \in \mathbb{R}, \quad l = 0, \ldots, n-1.$$

3. Bestimme die Koeffizienten der Cosinus-Transformation aus

$$\mathcal{C}_n f(l) = \frac{c(l)n}{2} \exp\big(\frac{-\pi i l}{2n}\big)\mathcal{F}_{2n} f(l) \in \mathbb{R}, \quad l = 0, \ldots, n-1.$$

Die Rücktransformation erfolgt mit ganz analogen Formeln.

14.6 Zweidimensionale Transformationen

Sei $f \in \mathbb{C}^{n \times n}$ ein zweidimensionaler Datensatz,

$$f(j,k), \quad j,k = 0,\ldots,n-1.$$

14.6 Zweidimensionale Transformationen

Sei $f \in \mathbb{C}^{n \times n}$ ein zweidimensionaler Datensatz,

$$f(j,k), \quad j,k = 0,\ldots,n-1.$$

Die Wellenfunktionen

$$\phi_{l.m}(j,k) = \exp(2\pi i (jl + km)/n) = \phi_l(j)\,\phi_m(k), \quad l,m = 0,\ldots,n{-}1,$$

14.6 Zweidimensionale Transformationen

Sei $f \in \mathbb{C}^{n\times n}$ ein zweidimensionaler Datensatz,

$$f(j,k), \quad j,k = 0,\dots,n-1.$$

Die Wellenfunktionen

$$\phi_{l.m}(j,k) = \exp(2\pi i(jl+km)/n) = \phi_l(j)\,\phi_m(k), \quad l,m = 0,\dots,n{-}1,$$

bilden ein Orthogonalsystem des $\mathbb{C}^{n\times n}$ bezüglich des Produkts

$$(f,g)_{n\times n} = \sum_{j,k=0}^{n-1} f(j,k)\,\overline{g(j,k)},$$

14.6 Zweidimensionale Transformationen

Sei $f \in \mathbb{C}^{n\times n}$ ein zweidimensionaler Datensatz,

$$f(j,k), \quad j,k = 0,\ldots,n-1.$$

Die Wellenfunktionen

$$\phi_{l.m}(j,k) = \exp(2\pi i(jl+km)/n) = \phi_l(j)\,\phi_m(k), \quad l,m = 0,\ldots,n-1,$$

bilden ein Orthogonalsystem des $\mathbb{C}^{n\times n}$ bezüglich des Produkts

$$(f,g)_{n\times n} = \sum_{j,k=0}^{n-1} f(j,k)\,\overline{g(j,k)},$$

nämlich

$$(\phi_{l,m},\phi_{n,o})_{n\times n} = n^2\delta_{(l,m),(n,o)}.$$

Daraus erhalten wir die zweidimensionale Fourier-Transformation

$$\mathcal{F}_{n\times n} f(l,m) = \frac{1}{n^2} \sum_{j,k=0}^{n-1} f(j,k) \exp(-2\pi i(jl+km)/n)$$

Daraus erhalten wir die zweidimensionale Fourier-Transformation

$$\mathcal{F}_{n\times n} f(l,m) = \frac{1}{n^2} \sum_{j,k=0}^{n-1} f(j,k) \exp(-2\pi i (jl+km)/n)$$

$$= \frac{1}{n^2} (f, \phi_{l,m})_{n\times n}.$$

Daraus erhalten wir die zweidimensionale Fourier-Transformation

$$\begin{aligned}\mathcal{F}_{n\times n} f(l,m) &= \frac{1}{n^2} \sum_{j,k=0}^{n-1} f(j,k) \exp(-2\pi i(jl+km)/n) \\ &= \frac{1}{n^2} (f, \phi_{l,m})_{n\times n}.\end{aligned}$$

Umkehrtransformation:

$$f(j,k) = \sum_{l,m=0}^{n-1} \mathcal{F}_{n\times n} f(l,m) \exp(2\pi i(jl+km)/n).$$

Zweidimensionale Fourier-Transformationen

Aufgrund der Tensorproduktstruktur der Wellenfunktionen und der Fourier-Transformation kommt gegenüber dem eindimensionalen Fall nichts wirklich Neues hinzu.

Aufgrund der Tensorproduktstruktur der Wellenfunktionen und der Fourier-Transformation kommt gegenüber dem eindimensionalen Fall nichts wirklich Neues hinzu.

Allerdings ist die Unterscheidung zwischen langen und kurzen Wellen nicht mehr so leicht möglich, weil Wellen in einer Komponente lang und in der anderen kurz sein können.

Aufgrund der Tensorproduktstruktur der Wellenfunktionen und der Fourier-Transformation kommt gegenüber dem eindimensionalen Fall nichts wirklich Neues hinzu.

Allerdings ist die Unterscheidung zwischen langen und kurzen Wellen nicht mehr so leicht möglich, weil Wellen in einer Komponente lang und in der anderen kurz sein können.

Mit

$$\mathcal{F}_{n\times n} f(l,m) = \frac{1}{n^2} \sum_{k=0}^{n-1} \exp(-2\pi ikm/n) \sum_{j=0}^{n-1} f(j,k) \exp(-2\pi ijl/n)$$

lässt sich die zweidimensionale Transformation auf die eindimensionale zurückführen:

Zweidimensionale Fourier-Transformationen

$$\mathcal{F}_{n\times n} f(l,m) = \frac{1}{n^2} \sum_{k=0}^{n-1} \exp(-2\pi i k m/n) \sum_{j=0}^{n-1} f(j,k) \exp(-2\pi i j l/n)$$

$$\mathcal{F}_{n\times n} f(l,m) = \frac{1}{n^2} \sum_{k=0}^{n-1} \exp(-2\pi i k m/n) \sum_{j=0}^{n-1} f(j,k) \exp(-2\pi i j l/n)$$

Wir führen zunächst für jedes k eine eindimensionale Transformation bezüglich j durch.

$$\mathcal{F}_{n\times n}f(l,m) = \frac{1}{n^2}\sum_{k=0}^{n-1}\exp(-2\pi ikm/n)\sum_{j=0}^{n-1}f(j,k)\exp(-2\pi ijl/n)$$

Wir führen zunächst für jedes k eine eindimensionale Transformation bezüglich j durch.

Die obige Formel zeigt, dass die auf diese Weise erhaltene Funktion $\tilde{f}(l,k)$ für jedes l bezüglich k transformiert werden muss.

Zweidimensionale Fourier-Transformationen

$$\mathcal{F}_{n\times n}f(l,m) = \frac{1}{n^2}\sum_{k=0}^{n-1}\exp(-2\pi ikm/n)\sum_{j=0}^{n-1}f(j,k)\exp(-2\pi ijl/n)$$

Wir führen zunächst für jedes k eine eindimensionale Transformation bezüglich j durch.

Die obige Formel zeigt, dass die auf diese Weise erhaltene Funktion $\tilde{f}(l,k)$ für jedes l bezüglich k transformiert werden muss.

Der Aufwand für dieses Verfahren ist damit $O(n^2(\log n)^2)$.

Die zweidimensionale Cosinus-Transformation ist analog zum eindimensionalen Fall durch Bildung der Tensorprodukte definiert,

$$\begin{aligned} &\mathcal{C}_{n\times n} f(l,m) \\ &= c(l)c(m) \sum_{j,k=0}^{n-1} f(j,k) \cos\left(\frac{\pi(2j+1)l}{2n}\right) \cos\left(\frac{\pi(2k+1)m}{2n}\right), \end{aligned}$$

$c(l)$ wie oben.

Zweidimensionale Cosinus-Transformation

Die zweidimensionale Cosinus-Transformation ist analog zum eindimensionalen Fall durch Bildung der Tensorprodukte definiert,

$$
\begin{aligned}
&\mathcal{C}_{n\times n} f(l,m) \\
&= c(l)c(m) \sum_{j,k=0}^{n-1} f(j,k) \cos\left(\frac{\pi(2j+1)l}{2n}\right) \cos\left(\frac{\pi(2k+1)m}{2n}\right),
\end{aligned}
$$

$c(l)$ wie oben.

Der Zusammenhang zwischen Fourier-Transformation für den gespiegelten Datensatz und Cosinus-Transformation bleibt der gleiche.

14.7 Bildkompression

Ein monochromes Bild ist ein Datensatz der Form

$$f(j,k), \quad j,k = 0,\dots,n-1.$$

14.7 Bildkompression

Ein monochromes Bild ist ein Datensatz der Form

$$f(j,k), \quad j,k = 0,\dots,n-1.$$

$f(j,k)$ ist der Grauton des Pixels (j,k).

14.7 Bildkompression

Ein monochromes Bild ist ein Datensatz der Form

$$f(j,k), \quad j,k = 0,\dots,n-1.$$

$f(j,k)$ ist der Grauton des Pixels (j,k).

Da für die Speicherung von $f(j,k)$ jeweils ein Byte verwendet wird, entspricht $f = 0$ schwarz und $f = 255$ weiß.

14.7 Bildkompression

Ein monochromes Bild ist ein Datensatz der Form

$$f(j,k), \quad j,k = 0,\ldots,n-1.$$

$f(j,k)$ ist der Grauton des Pixels (j,k).

Da für die Speicherung von $f(j,k)$ jeweils ein Byte verwendet wird, entspricht $f = 0$ schwarz und $f = 255$ weiß.

Zur Kompression wird das Bild zunächst in Teilbilder der Größe 8×8 oder 16×16 aufgeteilt.

14.7 Bildkompression

Ein monochromes Bild ist ein Datensatz der Form

$$f(j,k), \quad j,k = 0,\ldots,n-1.$$

$f(j,k)$ ist der Grauton des Pixels (j,k).

Da für die Speicherung von $f(j,k)$ jeweils ein Byte verwendet wird, entspricht $f = 0$ schwarz und $f = 255$ weiß.

Zur Kompression wird das Bild zunächst in Teilbilder der Größe 8×8 oder 16×16 aufgeteilt.

Da die JPEG-Kompression 8×8 verwendet, wollen wir hier nur diesen Fall weiterverfolgen.

JPEG-Standard

Auf jedem Teilbild wird eine Cosinus-Transformation durchgeführt und das Ergebnis als Matrix festgehalten.

JPEG-Standard

Auf jedem Teilbild wird eine Cosinus-Transformation durchgeführt und das Ergebnis als Matrix festgehalten.

Mit $\mathcal{C}f = \mathcal{C}_{8\times 8}f$ erhalten wir

$$\begin{pmatrix} \mathcal{C}f(0,0) & \dots & \mathcal{C}f(7,0) \\ \vdots & & \vdots \\ \mathcal{C}f(0,7) & \dots & \mathcal{C}f(7,7) \end{pmatrix}$$

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

JPEG-Standard

Das rechte Schema zeigt, wie die Koeffizienten auf einem eindimensionalen Feld abgespeichert werden. Abgesehen von Rundungsfehlern kann aus den Informationen zu diesem Zeitpunkt das vollständige Bild rekonstruiert werden.

Das rechte Schema zeigt, wie die Koeffizienten auf einem eindimensionalen Feld abgespeichert werden. Abgesehen von Rundungsfehlern kann aus den Informationen zu diesem Zeitpunkt das vollständige Bild rekonstruiert werden.

Die Kompression erfolgt nun dadurch, dass kleine und/oder hochfrequente Komponenten in der Matrix zu Null gesetzt oder grob gerundet werden.

Das rechte Schema zeigt, wie die Koeffizienten auf einem eindimensionalen Feld abgespeichert werden. Abgesehen von Rundungsfehlern kann aus den Informationen zu diesem Zeitpunkt das vollständige Bild rekonstruiert werden.

Die Kompression erfolgt nun dadurch, dass kleine und/oder hochfrequente Komponenten in der Matrix zu Null gesetzt oder grob gerundet werden.

Im einfachsten Fall werden im rechten Schema nur die ersten K Komponenten berücksichtigt, das sind in etwa die K mit den längsten Wellenlängen.

JPEG-Standard

Im JPEG-Standard wird von den Grautönen die Zahl 128 abgezogen, was allerdings nur die Komponente $(0, 0)$ der Cosinus-Transformation verändert.

Im JPEG-Standard wird von den Grautönen die Zahl 128 abgezogen, was allerdings nur die Komponente $(0, 0)$ der Cosinus-Transformation verändert.

Dann wird eine Cosinus-Transformation für $n = 8$ durchgeführt.

JPEG-Standard

Im JPEG-Standard wird von den Grautönen die Zahl 128 abgezogen, was allerdings nur die Komponente $(0, 0)$ der Cosinus-Transformation verändert.

Dann wird eine Cosinus-Transformation für $n = 8$ durchgeführt.

Jede Komponente der obigen Matrix wird nun durch eine Zahl geteilt, deren Größe von der zugrundeliegenden Frequenz abhängt, aber nicht Teil des Standards ist.

Im JPEG-Standard wird von den Grautönen die Zahl 128 abgezogen, was allerdings nur die Komponente $(0,0)$ der Cosinus-Transformation verändert.

Dann wird eine Cosinus-Transformation für $n = 8$ durchgeführt.

Jede Komponente der obigen Matrix wird nun durch eine Zahl geteilt, deren Größe von der zugrundeliegenden Frequenz abhängt, aber nicht Teil des Standards ist.

Anschließend werden die auf diese Weise geteilten und auf ganze Zahlen gerundeten Einträge wie oben auf einem eindimensionalen Feld abgespeichert.

Im JPEG-Standard wird von den Grautönen die Zahl 128 abgezogen, was allerdings nur die Komponente $(0,0)$ der Cosinus-Transformation verändert.

Dann wird eine Cosinus-Transformation für $n = 8$ durchgeführt.

Jede Komponente der obigen Matrix wird nun durch eine Zahl geteilt, deren Größe von der zugrundeliegenden Frequenz abhängt, aber nicht Teil des Standards ist.

Anschließend werden die auf diese Weise geteilten und auf ganze Zahlen gerundeten Einträge wie oben auf einem eindimensionalen Feld abgespeichert.

Dieses Feld wird in der Regel viele kleine Zahlen enthalten sowie am Ende lauter Nullen und kann daher erfolgreich komprimiert werden.

JPEG-Standard

Empfohlen wird als Teiler

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Diese Zahlen wurden durch Experimente bestimmt. Es können aber auch größere oder kleinere Zahlen verwendet werden, so dass man den Grad der Komprimierung verändern kann.

Diese Zahlen wurden durch Experimente bestimmt. Es können aber auch größere oder kleinere Zahlen verwendet werden, so dass man den Grad der Komprimierung verändern kann.

Bei der Rücktransformation werden die abgespeicherten Zahlen zunächst mit diesen Einträgen multipliziert, anschließend die Rücktransformation durchgeführt.