5.2. Die Fast Fourier-Transformation

5.2.1. Rekursive Formulierung der IDFT:

Die IDFT besteht aus n Summen mit n Summanden, die aus Produkten mit n-ten Einheitswurzeln bestehen.

Idee:
Zerlege die Summen geschickt in zwei Teilsummen halber Länge, aus denen sich die ursprünglichen Summen leicht gewinnen lassen.
Dazu notwendig: $n=2m$ gerade! Oder $m=n/2$.

Aufteilung in Summanden zu geradem und ungeradem Index

Für j=0,1,...,m-1 erhält man die erste Hälfte der Komponenten

$$v_j = \sum_{k=0}^{n-1} c_k \cdot \exp\left(\frac{2\pi i j k}{n}\right) =$$

$$= \sum_{k=0}^{n/2-1} c_{2k} \cdot \exp\left(\frac{2\pi i j 2k}{n}\right) + \sum_{k=0}^{n/2-1} c_{2k+1} \cdot \exp\left(\frac{2\pi i j (2k+1)}{n}\right)$$

$$= \sum_{k=0}^{m-1} c_{2k} \cdot \exp\left(\frac{2\pi i j k}{m}\right) + \omega^j \cdot \sum_{k=0}^{m-1} c_{2k+1} \cdot \exp\left(\frac{2\pi i j k}{m}\right)$$

Daher erhält man also die erste Hälfte der Komponenten von v aus zwei IDFT's halber Länge m, einmal angewendet auf den Vektor der geradzahligen Indizes von c, und einmal auf den Vektor der ungeradzahligen Indizes von c.

Entsprechend muss noch die zweite Hälfte von v bestimmt werden:

$$v_{m+j} =$$

$$= \sum_{k=0}^{m-1} c_{2k} \cdot \exp\left(\frac{2i\pi(j+m)k}{m}\right) + \omega^{j+m} \cdot \sum_{k=0}^{m-1} c_{2k+1} \cdot \exp\left(\frac{2i\pi(j+m)k}{m}\right)$$

$$= \sum_{k=0}^{m-1} c_{2k} \cdot \exp\left(\frac{2ijk\pi}{m}\right) - \omega^{j} \cdot \sum_{k=0}^{m-1} c_{2k+1} \cdot \exp\left(\frac{2ijk\pi}{m}\right)$$

da $\omega^m = \exp(2im\pi/n) = \exp(i\pi) = \cos(\pi) + i \cdot \sin(\pi) = -1$

und $$\exp\left(\frac{2imk\pi}{m}\right) = \exp(2ik\pi) = 1^k = 1$$

Daher ergibt sich die zweite Hälfte des Vektors v aus denselben Fouriertransformierten halber Länge wie vorher, diesmal aber aus

der Differenz.
Damit lässt sich also

$$\begin{pmatrix} v_0 & \cdots & v_{n-1} \end{pmatrix} = IDFT\begin{pmatrix} c_0 & \cdots & c_{n-1} \end{pmatrix}$$

zurückführen auf

$$\begin{pmatrix} v_0^{(g)} & \cdots & v_{m-1}^{(g)} \end{pmatrix} = IDFT\begin{pmatrix} c_0 & c_2 & \cdots & c_{n-2} \end{pmatrix} \quad \text{und}$$

$$\begin{pmatrix} v_0^{(u)} & \cdots & v_{m-1}^{(u)} \end{pmatrix} = IDFT\begin{pmatrix} c_1 & c_3 & \cdots & c_{n-1} \end{pmatrix}$$

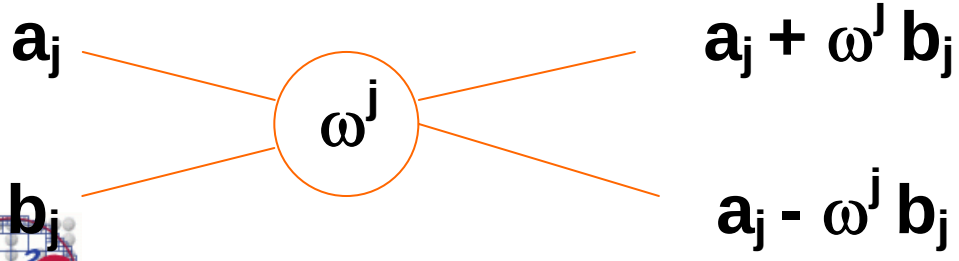
mit der Kombination für j=0,1,...,m-1:

$$v_j = v_j^{(g)} + \omega^j \cdot v_j^{(u)} \qquad und \qquad v_{j+m} = v_j^{(g)} - \omega^j \cdot v_j^{(u)}$$

Gerade/ungerade in c → erste Hälfte/zweite Hälfte in v
Entsprechend wird die IDFT der halb so langen Vektoren wieder

auf jeweils zwei IDFT (viertelter Länge) zurückgeführt.
Anstelle einer IDFT Länge n sind also nun vier IDFT's der Länge n/4 zu berechnen.
Allgemein dazu notwendig: $n=2^p$ ist Zweierpotenz;
Dann kann dieses Verfahren rekursiv durchgeführt werden, bis man *eine IDFT der Länge n* ausgedrückt hat durch
n IDFT's der Länge 1.

Grundidee:
Rekursive Aufteilung des aktuellen Vektors in geraden und ungeraden Anteil; falls die Ergebnisse der beiden IDFT's halber Länge vorliegen, werden die beiden Teile zum Ergebnis doppelter Länge kombiniert mittels



Rekursives Programm für IFFT:

FUNCTION($v_0,\ldots,v_{n-1}$) = IDFT($c_0,\ldots,c_{n-1},n$)

IF n==1

  $v_0 = c_0$;

ELSE

  m=n/2 ;

  ($g_0,\ldots, g_{m-1}$) = IDFT($c_0, c_2,\ldots, c_{n-2},m$) ;

  ($u_0,\ldots, u_{m-1}$) = IDFT($c_1, c_3,\ldots, c_{n-1},m$) ;

  $\omega = \exp(2i\pi/n)$;

  FOR j=0:m-1

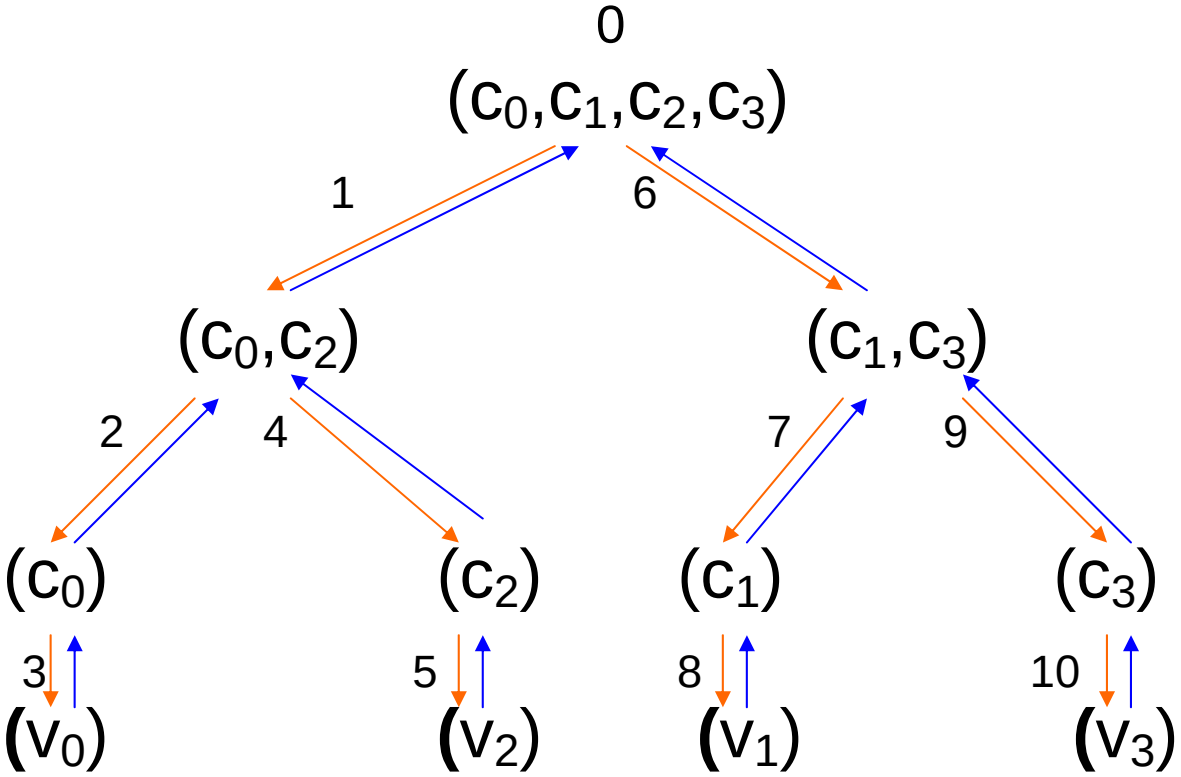
    $v_j = g_j + \omega^j u_j$;

    $v_{j+m} = g_j - \omega^j u_j$;

  END

END

5.2.2 Genauere Analyse des rekursiven Programms: Aufrufabfolge als binärer Baum (hier n=4):



Ziffern 0 bis 10 bezeichnen die unterschiedlichen rekursiven Aufrufe des Programms IDFT mit neuen Parametern (roter Pfeil: Aufruf, blauer Pfeil: Werterückgabe und Ende).

Insgesamt 3n-1 Programmaufrufe IDFT.

Die Aufrufe auf der untersten Ebene (3,5,8,10) sind trivial, da mit Vektoren der Länge 1 als Eingabe.

Kosten für $n=2^p$ ($p=\log(n)$):

Auf jeder Ebene sind jeweils 2^k Vektoren der Länge 2^{n-k} zu bearbeiten (umsortieren und zusammenkombinieren):

	Kosten
Ebene 1: 2 Vektoren der Länge n/2	O(*n*)
Ebene 2: 4 Vektoren der Länge n/4	O(*n*)
....	
Ebene p: n Vektoren der Länge 1	O(*n*)

Gesamtkosten daher

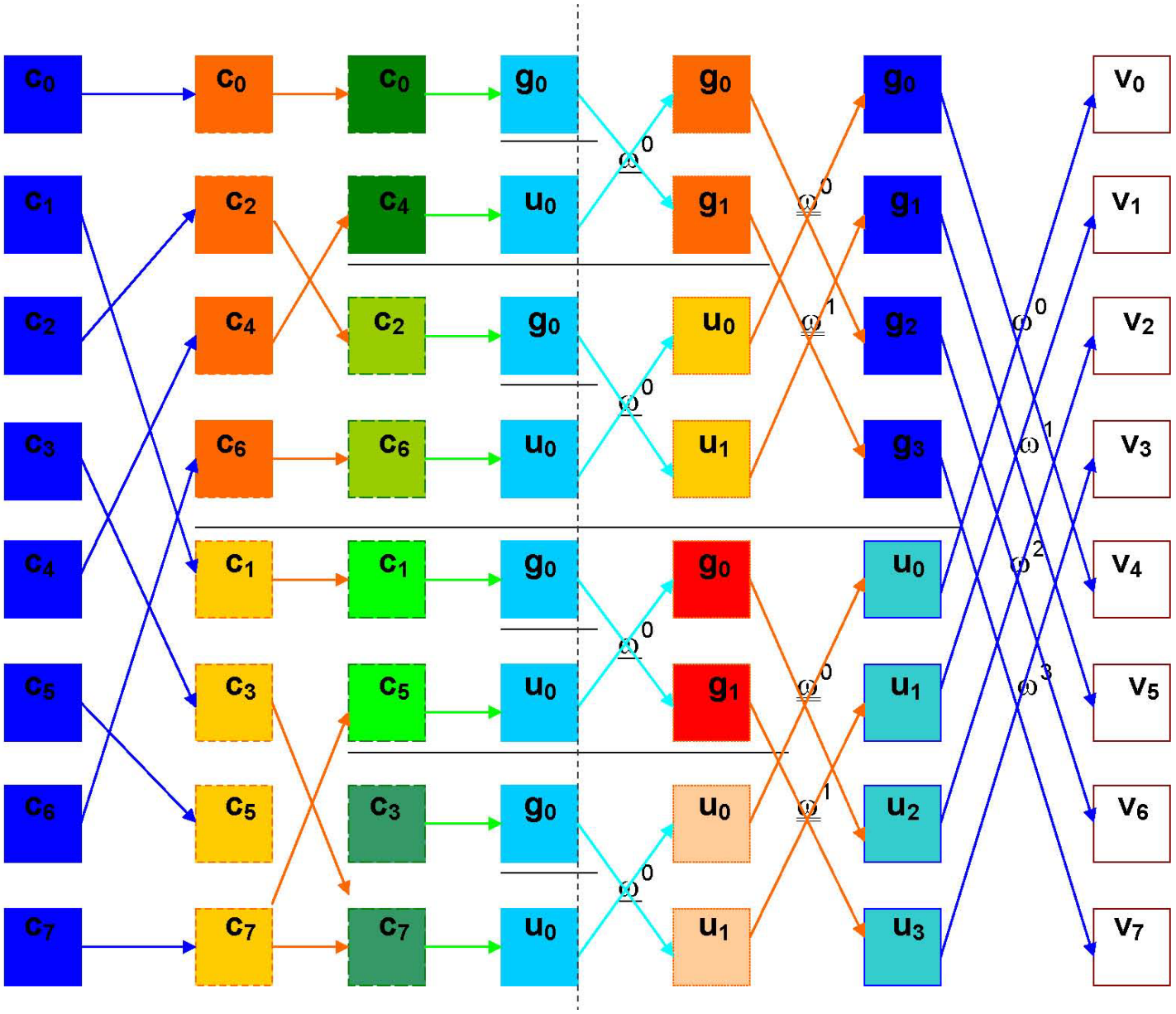
$$\mathbf{O(p*n) = O(p*2^p) = O(\log_2(n)*n)}$$

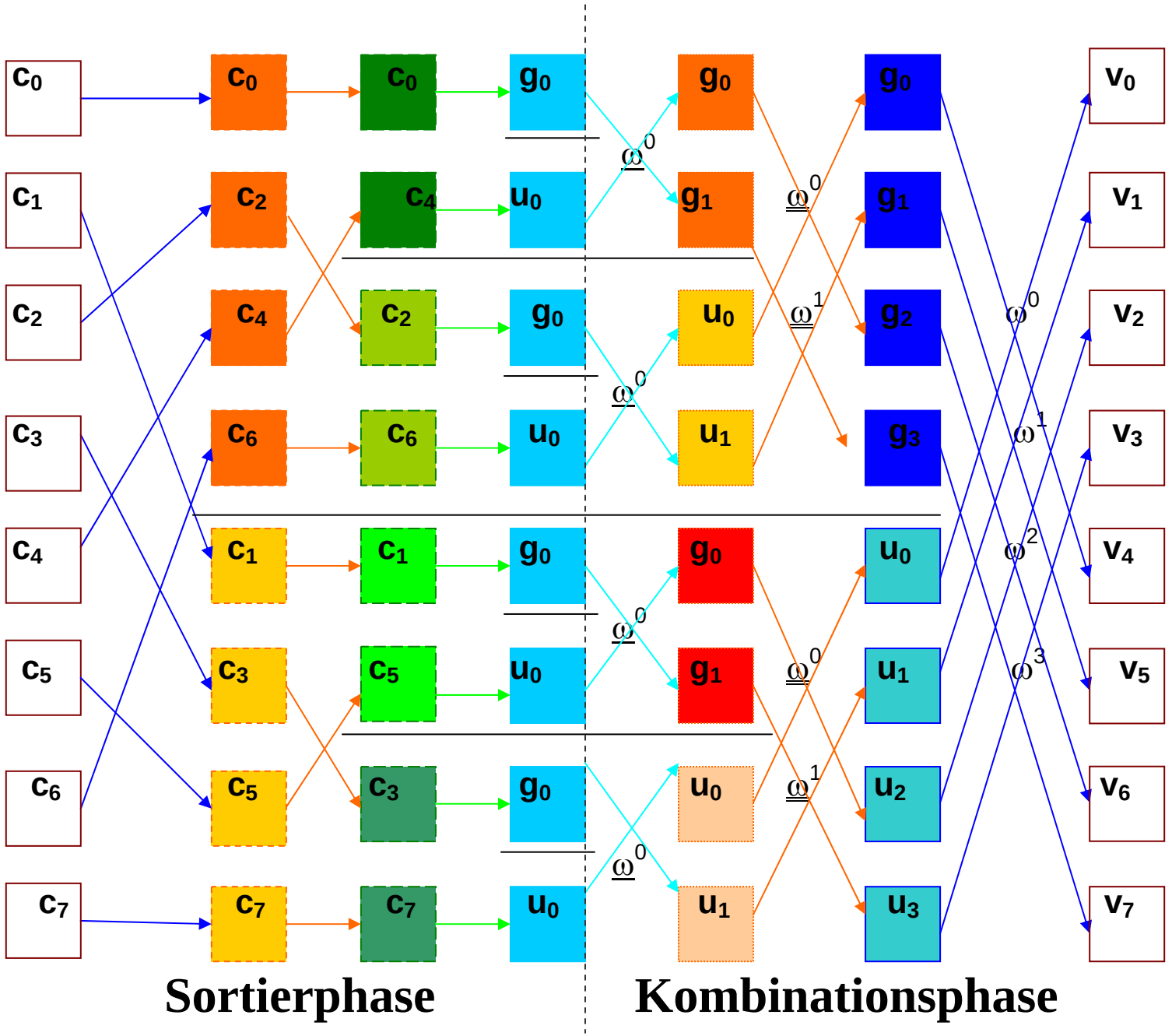
anstatt $\mathbf{O(n^2)}$, wie sonst für Matrixmultiplikation.

Rekursives Programm zwar in O(pn) sehr schnell, aber großer Overhead für Aufrufe:

Es muss dynamisch laufend neuer Speicherplatz angelegt werden.

Ziel ist es im Folgenden, die Rekursion besser durch einen einfachen Loop darzustellen.


c0
c1
c2
c3
c4
c5
c6
c7
c0
c2
c4
c6
c1
c3
c5
c7
c0
c4
c2
c6
c1
c5
c3
c7
g0
u0
g0
u0
g0
u0
g0
u0
g0
g1
u0
u1
g0
g1
u0
u1
g0
g1
g2
g3
u0
u1
u2
u3
v0
v1
v2
v3
v4
v5
v6
v7
ω0
ω0
ω0
ω1
ω0
ω1
ω0
ω1
ω2
ω3


c0
c1
c2
c3
c4
c5
c6
c7
g0
g1
g2
g3
u0
u1
u2
u3
v0
v1
v2
v3
v4
v5
v6
v7
ω0
ω1
ω2
ω3
Sortierphase
Kombinationsphase

Passendes ω: $\underline{\omega}$ **zu n=2: exp(iπ), j=0;** $\underline{\underline{\omega}}$ **zu n=4: exp(iπ/2), j=0,1**

ω zu n=8: exp(iπ/4), j=0,1,2,3

Der Faktor, der im Butterfly bei einem Kombinationsschritt verwendet werden muss, hängt ab von der aktuellen Länge der zu kombinierenden Vektoren und vom Index j, der die Stelle der Komponente angibt, die gerade kombiniert werden.

In jeder Spalte werden gleiche Komponenten von g und u zu Index j zu neuem v kombiniert (das dann auch wieder g oder u für die nächsten Spalte ist)

Idee:
Schreibe zwei einfache Loop-Programme, eines für die *Sortierphase* und eines für die *Kombinationsphase*.

5.2.3. Sortierphase

Betrachte Indizes k von Eingabevektor c_k im Binärsystem:

$$k = \left(b_{p-1} \ldots b_1 b_0\right)_2 \quad mit \quad n = 2^p$$

In jedem Schritt wird nach gerade/ungerade sortiert, also im ersten Schritt nach der **letzten** Binär-Stelle b_0:
Alle Komponenten, mit $b_0=0$ kommen nach vorne,
alle mit $b_0=1$ nach hinten.

Im zweiten Schritt wird wieder nach gerade/ungerade sortiert in den beiden Teilvektoren, d.h.
in den zwei Teilvektoren $b_1=0$ nach vorne, $b_1=1$ nach hinten,
also die mit Zweierstelle 1 nach hinten

usw.

Dies entspricht einer Sortierung von hinten!
Bei der üblichen Sortierung stehen Zahlen mit $b_{p-1}=1$ hinten und mit $b_{p-1} = 0$ vorne. Danach wird bei der üblichen Sortierung dasselbe bezüglich b_{p-2} wiederholt. ...
So entsteht die üblich sortierte Reihenfolge.

In unserer neuen Sortierphase passiert dasselbe, aber von hinten her, beginnend mit b_0.

Insgesamt wird durch die rekursiven Aufrufe also die Folge 0,1,...,n-1 aufsteigend sortiert, aber von hinten her! Daher ist die Bitreihenfolge nach der Permutierung dann genau revertiert.

Wir bezeichnen mit $\pi(k)=\pi((b_{p-1}...b_1b_0)_2)=(b_0...b_{p-1})_2$
das Ergebnis dieser Vertauschungen → Bitreversal.

	k		→	π(k)
0	000	000	000	0
1	001	010	100	4
2	010	100	010	2
3	011	110	110	6
4	100	001	001	1
5	101	011	101	5
6	110	101	011	3
7	111	111	111	7

Letzte Spalte mit Bitwerte in ‚reverser' Reihenfolge, sortiert nach Stellen in der Binärdarstellung in umgekehrter Reihenfolge.

Beispiel:
p=3, n=8:

$$k = 5 = (101)_2 \rightarrow \pi(k) = (101)_2 = 5$$
$$k = 3 = (011)_2 \rightarrow \pi(k) = (110)_2 = 6$$
$$k = 1 = (001)_2 \rightarrow \pi(k) = (100)_2 = 4$$

Programmidee für Bit-Reversal:
Zähle für k=0,1,...,n-1 beginnend mit $\pi(0)=0$ mit $\pi(k)$ ‚hoch', also berechne aus $\pi(k-1)$ jeweils das nächste $\pi(k)$.

Sei $k-1 = (b_{p-1}b_{p-2} \ldots 011\ldots11)_2$, und daher
$\pi(k-1) = (11\ldots110 \ldots b_{p-2}b_{p-1})_2$;

Daher ist dann $k = (b_{p-1}b_{p-2} \ldots 100\ldots00)_2$, und daher
$\pi(k) = (00\ldots001 \ldots b_{p-2}b_{p-1})_2$;

Um aus $\pi(k-1)$ also $\pi(k)$ zu erhalten, muss das erste (von links) zusammenhängende Einserpaket in $\pi(k-1)$ durch ein Nullerpaket ersetzt werden,
und die in $\pi(k-1)$ daran anschließende erste 0 durch eine 1 in $\pi(k)$ ersetzt werden.

Der Rest bleibt unverändert.

Programm Bitreversal:

```
π(0) = 0 ;
FOR k = 1:n-1
        π(k) = π(k-1) ;
        FOR j = 1 : p
                IF j-tes Bit von π(k) ist 0:
                    Setze j-tes Bit von π(k) gleich 1  ; % 0→1
                    break ;                  %k fertig
                ELSE
                    Setze j-tes Bit von π(k) gleich 0 ;   % 1→0
                ENDIF
        ENDFOR
ENDFOR
```

Für jedes k:

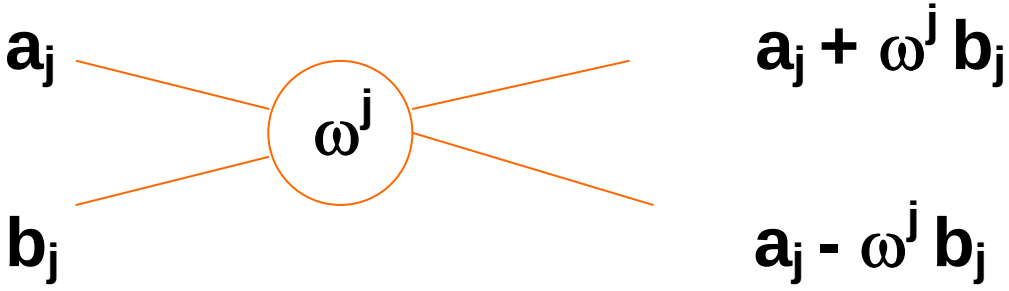
Frage von links her ab, ob aktuelle Stelle zu erstem Einserpaket gehört:

- Wenn nein, dann ersetze durch 1, und die Bearbeitung für dieses k ist fertig; k wird um 1 erhöht

- Wenn ja, dann ersetze diese 1 durch 0, und gehe zur nächsten Stelle eine Position nach rechts

5.2.4. Kombinationsphase

Die umsortierten Komponenten sind nun paarweise in der Form (als Butterfly)



zu kombinieren, und zwar mit wechselndem $\omega^j=\exp(2ij\pi/n)$, je nach Länge n des durch die Kombination entstehenden Gesamtvektors und je nach Stelle j in diesem Vektor.

ω ist dabei also abhängig von der Spalte, in dem Tableau (rechte Spalte bezieht sich auf Maximallänge n, usw.).

j ist davon abhängig, die wievielte Komponente in den beiden Vektoren gerade mit dem Butterfly kombiniert wird.

Der Abstand zwischen zwei zu kombinierenden Einträgen ist die aktuelle Vektorlänge.

Im Folgenden bezeichnet

k: die jeweilige zu bearbeitende Spalte.

$\omega_j :\Leftarrow \omega^j$

j: Komponente j in den zu kombinierenden Teilvektoren

s: die in Spalte k auftretenden Vektor-Paar-Kombinationen

Programm Kombinationsphase

```
r = 1 ;
FOR k = 1:p
        m = r ;
        r = 2m ;
        FOR j = 0:m-1
            ω_j= cos(πj/m) + i sin(πj/m);
            FOR s = 0:r:n-r
                g = v_{j+s} ;
                h = ω_j·v_{j+s+m} ;
                v_{j+s} = g + h ;
                v_{j+s+m} = g - h ;
            ENDFOR
        ENDFOR
ENDFOR
```

5.2.5. Kosten:

Kosten für IDFT der Länge 2n =
(2 * Kosten für IDFT der Länge n) + a*n

also

$$T_{2n} = 2*T_n + a*n$$

Setze $f_p := T_{2^p} = T_n$, $n=2^p$ oder $p=\log_2(n)$

Dann gilt

$f_{p+1} = 2*f_p + a*n = 2*f_p + a*2^p$, * x^p , $\sum$

$$\sum_{p=0}^{\infty} f_{p+1} x^p = 2 \cdot \underbrace{\sum_{p=0}^{\infty} f_p x^p}_{\mathbf{F(x)}} + a \cdot \sum_{p=0}^{\infty} 2^p x^p$$

$(F(x)-f_0) / x = 2*F(x) + a / (1-2x)$

$F(x)*(1-2x) = ax / (1-2x) + f_0$

$F(x) = ax / (1-2x)^2 + f_0 / (1-2x)$

heißt ‘Erzeugende Funktion’ der Folge f_p .

Reihenentwicklung von F(x):

$$a \cdot \sum_{p=0}^{\infty} \frac{(p+1)!}{p!} 2^p x^{p+1} + f_0 \cdot \sum_{p=0}^{\infty} 2^p \cdot x^p =$$

$$= f_0 + \sum_{p=1}^{\infty} \left(a \frac{p!}{(p-1)!} 2^{p-1} + f_0 2^p \right) \cdot x^p =$$

$$= f_0 + \sum_{p=1}^{\infty} (pa + 2 f_0) 2^{p-1} x^p$$

und daher $f_p = (ap + 2f_0)2^{p-1}$ für p=1,2,... oder

$$T_n = T_{2^p} = (ap + 2f_0)2^{p-1} = \frac{a}{2} n \cdot \log_2(n) + f_0 n = O(n \cdot \log_2(n))$$

$f_0 = T_1$: Kosten für IDFT der Länge 1
a : Kosten für Operationen in einem Butterfly

Durchführbarkeit der Divide & Conquer – Idee ist nur möglich, wenn n in viele Faktoren zerfällt (z.B. wenn n eine Zweier-potenz ist).

Allgemein kann man bei einer Primfaktorzerlegung der Form $n=n_1 \ldots n_p$ eine Aufspaltung

in n_1 Teilsummen im ersten Schritt,

... ,

in n_p Teilsummen im letzten Schritt

ausführen.

Ist z.B. n durch drei teilbar, so lässt sich die IDFT-Summe durch drei IDFT's der Länge n/3 ausdrücken.

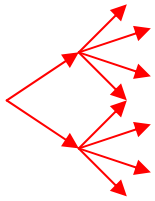
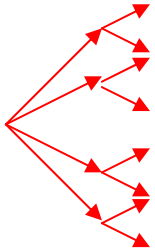
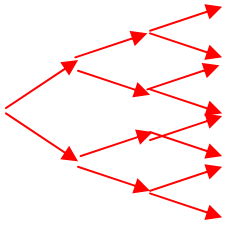
Ist n eine Primzahl → FFT geht so nicht!
Durch gruppentheoretische Überlegungen lässt sich dann aber dieser Fall auf die Berechnung der FFT zu n-1 und n-2 zurückführen, die dann keine Primzahl sind.

FFTW = Fastest Fourier Transform in the West:

Teste für CPU, Cache, usw. die verschiedenen Faktorzerlegungen von n.
Wähle diejenige mit schnellster Laufzeit aus und erstelle automatisch FFT-Routine.

n= 8 = 1*8 = 2*4 = 2*2*2 = 4*2 = 8*1

IDFT Länge 8 oder
IDFT Länge 2 gefolgt von IDFT 4 oder
IDFT Länge 2, IDFT Länge 2, IDFT Länge 2 oder
IDFT Länge 4 gefolgt von IDFT 2

Beispiel: FFT für $n=3^q$

$$v_j = \sum_{k=0}^{n} c_k e^{2\pi i jk/n} = \sum_{k=0}^{n/3-1} c_{3k} e^{2\pi i j 3k/n} +$$

$$+ \sum_{k=0}^{n/3-1} c_{3k+1} e^{2\pi i j(3k+1)/n} + \sum_{k=0}^{n/3-1} c_{3k+2} e^{2\pi i j(3k+2)/n} =$$

$$= F_0 + e^{2\pi i j/n} F_1 + e^{2\pi i 2j/n} F_2$$

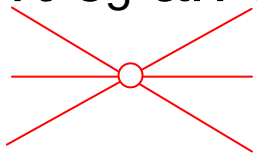
Dazu benötigt **IDFT(c_0 , c_3 , c_6 , ...)**
IDFT(c_1 , c_4 , c_7 , ...)
IDFT(c_2 , c_5 , c_8 , ...)

Sortieren modulo 3: Bitreversal in ternärer Darstellung
z.B. dreistellig für **$n = 3^3 = 27$:**

$5 = 1\cdot3+2\cdot1 = (012)_3 \rightarrow (210)_3 = 2\cdot9+1\cdot3=21$

In Sortierphase wandert c_5 an die Position c_{21} .

‚Dreier-Butterfly'



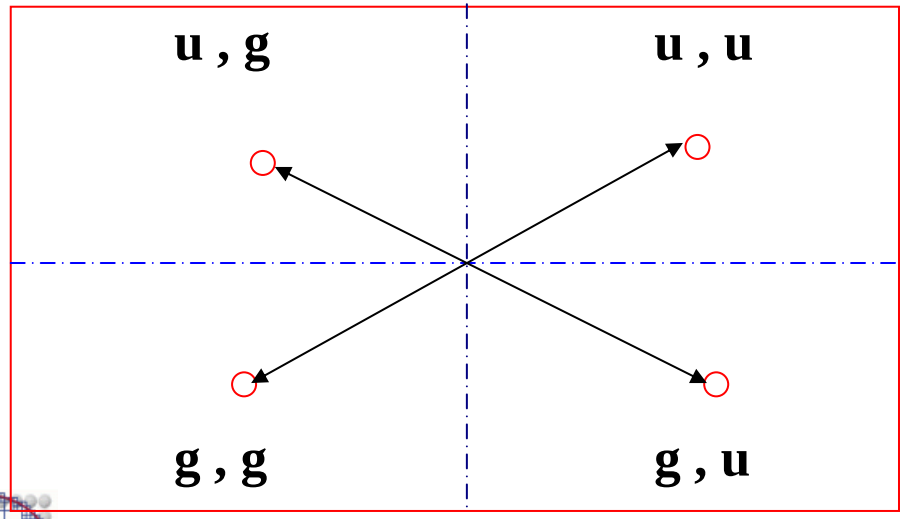
Beispiel: FFT in 2D

$$v_{j,s} = \sum_{k=0}^{N} \sum_{r=0}^{M} c_{k,r} e^{2\pi i jk/N} e^{2\pi i rs/M} =$$

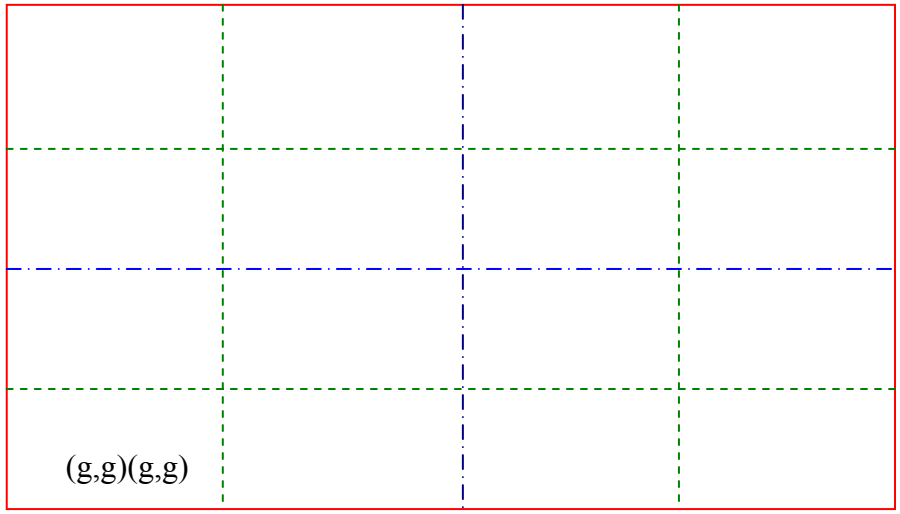
$$= \sum_{k=0}^{N/2-1} \sum_{r=0}^{M/2-1} c_{2k,2r} e^{2\pi i j 2k/N} e^{2\pi i 2rs/M} + c_{2k,2r+1} e^{2\pi i j 2k/N} e^{2\pi i (2r+1)s/M}$$

$$+ c_{2k+1,2r} e^{2\pi i j(2k+1)/N} e^{2\pi i 2rs/M} + c_{2k+1,2r+1} e^{2\pi i j(2k+1)/N} e^{2\pi i (2r+1)s/M} =$$

$$= F_{00} + e^{2\pi i s/M} F_{10} + e^{2\pi i j/N} F_{01} + e^{2\pi i j/N} e^{2\pi i s/M} F_{11}$$



Butterfly kombiniert 4 Komponenten zu 4 neuen Komponenten



Verallgemeinerung NFFT:
Schneller Algorithmus für „nonequispaced“ Stützstellen, also nicht die n-ten Einheitswurzeln.

5.3. Anwendungen

5.3.1. Fourierentwicklung:

Stückweise stetige, 2π-periodische (in $[-\pi,\pi]$) Funktion f(x) lässt sich als Fourier-Reihe darstellen (vgl. Taylor/Potenz-Reihe):

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos(kx) + b_k \sin(kx) \right)$$

$$\left\{ \equiv \sum_{k=-\infty}^{k=\infty} c_k e^{ikx} = \sum_{k=-\infty}^{k=\infty} c_k \left(e^{ix} \right)^k \right\}$$

a_k und b_k heißen Fourier-Koeffizienten von f(x) und berechnen sich (wegen der Orhogonalität der Funktionen cos(kx) und sin(kx)) aus

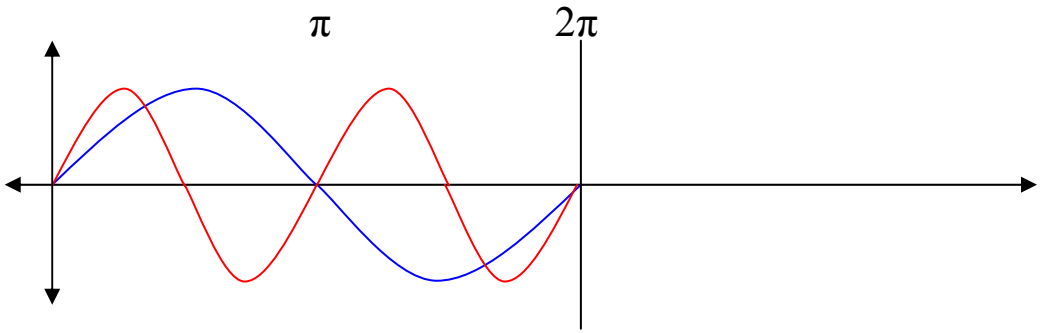
$$a_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)dx, \quad b_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(kx)dx$$

Sie geben die Größe der Anteile von Vielfachen der Grundfrequenz (1/ 2π) an, aus denen sich f zusammensetzt

Wellenzahl: k
Periode: $2\pi/k$
Frequenz: $k/2\pi$

Grundfrequenz für k=1

(a_k,b_k) messen Anteil von f zur Frequenz k / 2π,
(bzw. Periode 2π/k, oder Wellenzahl k).



Schwingung mit Wellenzahl k=1 und k=2

Wellenzahl k=2 entspricht Periode π, bzw. Frequenz 1/π

cos(kx), sin(kx) bilden Orthonormalsystem (Basis)!

Das trigonometrische Polynom

$$f_n(x) = \frac{a_0}{2} + \sum_{k=1}^{n} \left(a_k \cos(kx) + b_k \sin(kx) \right)$$

stellt die optimale Approximation an die Funktion f dar aus dem Vektorraum der trigonometrischen Polynome vom Grad n:

$$\| f_n(x) - f(x) \|_2^2 = \int_{-\pi}^{\pi} \left(f_n(x) - f(x) \right)^2 dx$$

ist minimal.

Näherungsweise Berechnung der Fourierkoeffizienten aus dem Integral mittels Trapezregel und äquidistanten Stützstellen

$$x_0 = -\pi, \qquad x_j = -\pi + j\frac{2\pi}{n}, \qquad x_n = \pi.$$

$$a_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)dx \approx$$

$$\approx \frac{1}{\pi}\cdot\frac{2\pi}{n}\cdot\left(\frac{1}{2}f(-\pi)\cos(-k\pi) + f(x_1)\cos(kx_1) + \ldots\right.$$

$$\left.\ldots + f(x_{n-1})\cos(kx_{n-1}) + \frac{1}{2}f(\pi)\cos(k\pi)\right) =$$

$$= \frac{2}{n}\sum_{j=0}^{n-1} f(x_j)\cos\left(k(-\pi + j\frac{2\pi}{n})\right) =$$

$$= \frac{2}{n}\cos(k\pi)\sum_{j=0}^{n-1} f(x_j)\cos\left(\frac{2\pi kj}{n}\right) + \frac{2}{n}\sin(k\pi)\sum_{j=0}^{n-1} f(x_j)\sin\left(\frac{2\pi kj}{n}\right)$$

Daher ergeben sich die a_k näherungsweise aus den Real- und Imginärteilen von

$$\sum_{j=0}^{n-1} f(x_j)\exp\left(\frac{2i\pi kj}{n}\right) = \sum_{j=0}^{n-1} f(x_j)\cos\left(\frac{2\pi kj}{n}\right) + i\cdot\sum_{j=0}^{n-1} f(x_j)\sin\left(\frac{2\pi kj}{n}\right)$$

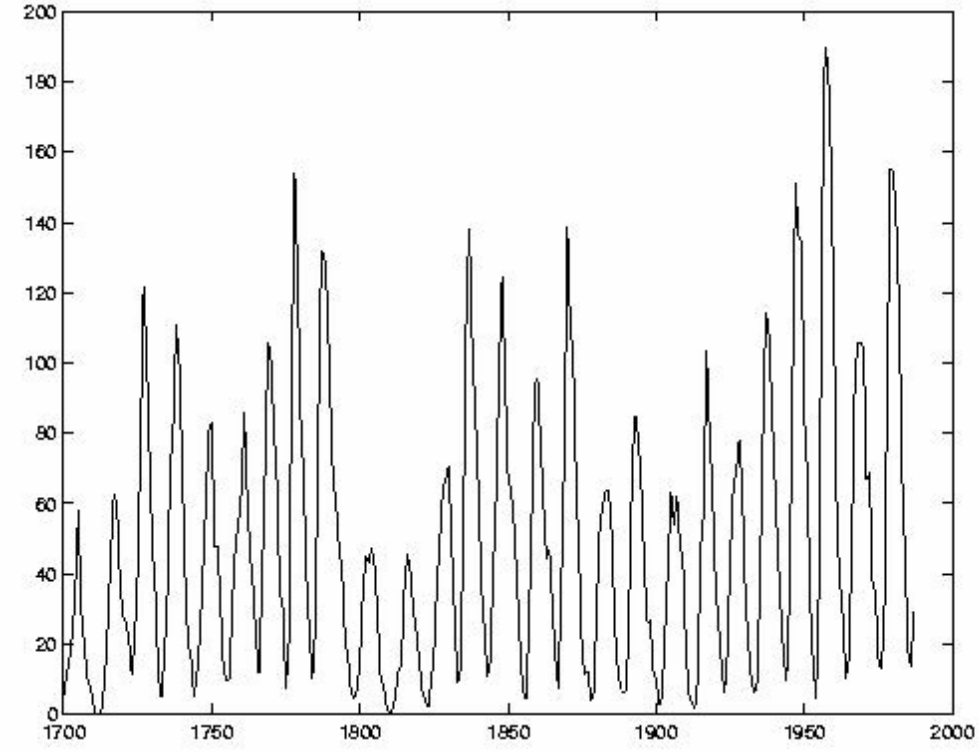
Diese Werte erhält man aus der IDFT angewendet auf die Funktionswerte an den Stützstellen.

Genauso lassen sich die b_k annähern.

Also können mittels DFT auf den Vektor der Funktionswerte die Frequenzanteile von f näherungsweise bestimmen werden.

5.3.2. MATLAB-Analyse der Sonnenaktivität:

Sonnenflecken treten alle 11 Jahre verstärkt auf.
Seit 1700 wird die jährliche Sonnenfleckenaktivität beschrieben durch die sog. Wolfer-Zahl (Größe und Anzahl der Flecken) (Rudolf Wolf ca. 1850).



Wolferzahl für die Jahre 1700 bis 2000

Wolferzahlen in Vektor v als Funktionswerte einer unbekannten periodischen Funktion g mit den Jahreszahlen als Stützstellen.

Gesamtbeobachtungszeitraum T besteht aus 300 Jahren.

Ersetze daher Intervall $[0,2\pi]$ durch das Intervall $[0,T]$ durch den Übergang von

$\exp(ikx) \rightarrow \exp(ikx*2\pi/300)$

Zeitintervall $\Delta = 1$ Jahr, in dem eine Wolferzahl bestimmt wurde;

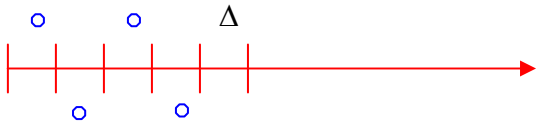
$N = T / \Delta$ ist Anzahl der Beobachtungsintervalle =

= Länge des Vektors v

= Anzahl der Stützstellen;

Wellenzahl wieder k , Periode T/k = 300J./k, Frequenz k/T;

Grundfrequenz also 1/T.



Es können nur periodische Vorgänge erfasst werden mit einer Periode > 2Δ = 2 Jahre, da bei kleineren Perioden die ‚Schwingung' feiner als die feinste Unterteilung (1 Jahr) wäre, und daher als solche nicht erkennbar ist.

Daher ist die größte beobachtbare Wellenzahl k gleich N/2=150 , entspricht der Frequenz (N/2)(1/T)=1/(2Δ), der sog. Nyquist-Frequenz

Berechne y = FFT(v)
y(0) ist die Gesamtsumme aller Wolferzahlen und wird gleich 0 gesetzt (enthält keine Angabe über Periodizität).

Die anderen Koeffizienten von y enthalten die Größe der verschiedenen Frequenzanteile von g (aber als komplexe Zahlen).

Berechne $p(k) = |y(k)|^2 \quad für \quad k = 1,..., N/2$,

also untersuche nur Frequenzanteile bis zur Nyquist-Wellenzahl k = (N/2).

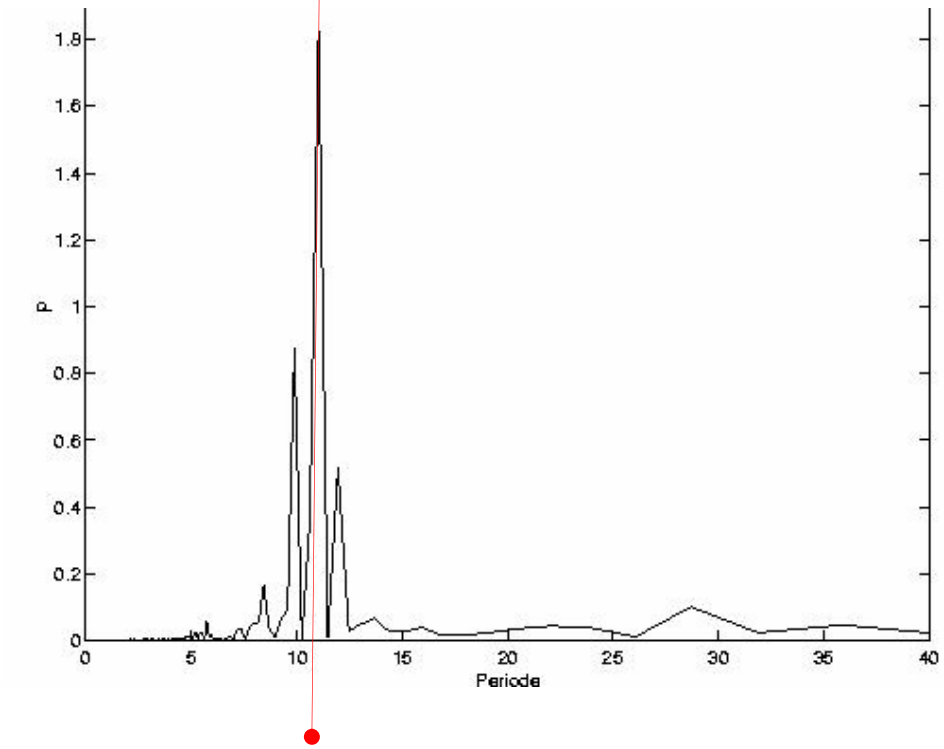
p(k) misst die Größe der k-ten Vielfachen der Grundfrequenz, also der Frequenz

$$f(k) = k / T = k / (\Delta N) \quad \text{für} \quad k=1,2,...,N/2$$

Anders ausgedrückt bestimmt p(k) das Gewicht der Periode

$$1 / f(k) = T / k$$ in der Funktion f.

Wir tragen nun den Vektor p auf gegen die dazugehörigen Perioden T/k, k=1,...,N/2



Betragsquadrate der Fourierkoeffizienten, aufgetragen über die dazugehörigen Perioden.
Wir können den Zyklus von 11 Jahren direkt ablesen.