14 Schnelle Fourier-Transformation

14.1 Die Lagrangesche Interpolationsaufgabe Mit $\mathbb{P}_n$ bezeichnen wir den Raum der komplexen Polynome $p : \mathbb{C} \to \mathbb{C}$ vom Grad $\leq n$. Gegeben seien $n+1$ verschiedene Stützstellen $x_j \in \mathbb{C}$, $j = 0, \ldots, n$, und $n+1$ nicht notwendig verschiedene Werte $y_0, \ldots, y_n \in \mathbb{C}$. In der *Lagrangeschen Interpolationsaufgabe* ist ein Polynom $p \in \mathbb{P}_n$ gesucht mit

$$p(x_j) = y_j, \quad j = 0, 1, \ldots, n. \tag{14.1}$$

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*.

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

$$l_j(x) = \frac{(x-x_0)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_0)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)}. \tag{14.2}$$

Es gilt dann $l_i(x_j) = \delta_{ij}$ und die Interpolationsaufgabe (14.1) wird gelöst durch

$$p(x) = \sum_{j=0}^{n} y_j l_j(x) \in \mathbb{P}_n. \tag{14.3}$$

Satz 14.1 *Die Interpolationsaufgabe (14.1) wird eindeutig gelöst durch das* Lagrangesche Interpolationspolynom *(14.3).*

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 $p_1 = p_2$. □

Ist $y_j = f(x_j)$ für ein Polynom $f \in \mathbb{P}_n$, so gilt für das Interpolationspolynom $p = f$, weil die Interpolationsaufgabe eindeutig lösbar ist. Polynome vom Grad $\leq n$ werden also in der Interpolation reproduziert.

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

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

für das Produkt

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

wobei die Konvention $a_j, b_j = 0$ für $j > n-1$ verwendet wurde. Da jeder Koeffizient des einen Polynoms mit jedem Koeffizienten des anderen Polynoms multipliziert wird, benötigen wir für die direkte Polynommultiplikation genau n^2 Multiplikationen und $O(n^2)$ Additionen.

Eine Alternative zur direkten Multiplikation ist die im vorigen Abschnitt besprochene 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. Da das Produkt $r(x) = p(x)q(x)$ im Raum $\mathbb{P}_{2n-2}$ liegt, genügt es, die Polynome p und q an $2n-1$ verschiedenen Stellen auszuwerten, die $2n-1$ Produkte $p(x_j)q(x_j)$ zu berechnen und anschließend das Interpolationspolynom zu diesen Produkten zu bestimmen. Da das Interpolationspolynom eindeutig bestimmt ist, wird durch diese Vorgehensweise das Produktpolynom reproduziert. Dieses Verfahren ist zunächst weniger effektiv als die direkte Methode, denn selbst wenn das Interpolationspolynom nach der raffiniertesten Methode bestimmt wird, werden dazu $O(n^2)$ Operationen benötigt.

Die Idee der schnellen Polynommultiplikation besteht in einer vorteilhaften Wahl der Stützstellen x_j, die eine simultane Auswertung von $p(x_j)$ gestattet.

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

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

Für diese gilt

$$\omega_n^k = \omega_n^{k+ln} \quad \forall l \in \mathbb{Z} \tag{14.4}$$

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

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, die nur noch den Grad $n/2 - 1$ besitzen, an den Stellen $\omega_n^{2k} = \omega_{n/2}^k$ ausgewertet werden, mit (14.4) sind dies $\omega_{n/2}^0, \ldots, \omega_{n/2}^{n/2-1}$. 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 $\omega_n^{2k} = \omega_n^{2k+2n/2}$.

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

procedure $FFT(n, p, \omega, a)$

bestimmt die Auswertung eines Polynoms vom Grade $n-1$ in den n Punkten $\omega_n^0, \ldots, \omega_n^{n-1}$. Die Zweierpotenz n gibt die Länge des Problems an, in der Inputvariablen p ist das auszuwertende Polynom als n-Vektor $p = (a_0, \ldots, a_{n-1})$ gespeichert. Ferner haben wir die Inputvariable $\omega = \omega_n$. Das Ergebnis ist der n-Vektor $a = (p(\omega^0), \ldots, p(\omega^{n-1}))$. Die schnelle Polynomauswertung kann damit folgendermaßen implementiert werden:

```
recursive subroutine FFT(n, p, ω, a)
  if n = 1 then
    a(0) = p(0)
  else
    n_2 = n/2
    p_g = (a_0, a_2, ..., a_{n-2})
    p_u = (a_1, a_3, ..., a_{n-1})
    call FFT(n_2, p_g, ω^2, g)
    call FFT(n_2, p_u, ω^2, u)
    do k = 0, n_2 - 1
      a(k) = g(k) + ω^k u(k)
      a(k + n_2) = g(k) - ω^k u(k)
    enddo
  endif
end
```

Zur Einsparung von Rechenoperationen wurde von der Beziehung $\omega_n^{k+n/2} = \omega_n^k \omega_n^{n/2} = -\omega_n^k$ Gebrauch gemacht.

Nun bestimmen wir die Anzahl der Multiplikationen $M(n)$ für diesen Algorithmus. Da die Zahlen ω^k einmal berechnet und anschließend abgespeichert werden können, fallen in der Do-Schleife nur die $n/2$ Multiplikationen $\omega^k u(k)$ an. Da FFT zweimal mit Inputlänge $n/2$ aufgerufen wird, genügt $M(n)$ der Rekursion

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

für $n = 2^l$ daher

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

Iterieren wir diese Beziehung l mal, so erhalten wir

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

Da im Then-Teil des obigen Programms keine Operationen anfallen, ist $M(1) = 0$ und daher

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

wobei mit log der Logarithmus zur Basis 2 bezeichnet wird. Da für die Anzahl der Addditionen eine analoge rekursive Beziehung gilt, erhalten wir für die Gesamtzahl an Operationen ebenfalls $O(n \log n)$.

Nun wenden wir uns dem Interpolationsproblem zu: Zu Daten $b_0, \ldots, b_{n-1}$ ist ein Polynom $p \in \mathbb{P}_{n-1}$ gesucht mit $p(\omega_n^k) = b_k$ für $k = 0, \ldots, n-1$. Wie bereits zu Anfang dieses Abschnitts 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. Zu $\alpha = (\alpha_0, \ldots, \alpha_{n-1})^T \in \mathbb{C}^n$ mit paarweise verschiedenen α_k definieren wir die zugehörige *Vandermondsche Matrix* als

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

Für ein Polynom

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

gilt dann

$$b := \big(p(\alpha_0), \ldots, p(\alpha_{n-1})\big)^T = V(\alpha)a, \quad a = (a_0, \ldots, a_{n-1})^T. \tag{14.5}$$

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. Für eine komplexe Zahl β schreiben wir $[\beta] = (\beta^0, \ldots, \beta^{n-1})^T$.

Satz 14.2 *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

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

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

$$\sum_{l=0}^{n-1} (\omega_n^{j-k})^l = \frac{\omega_n^{n(j-k)} - 1}{\omega_n^{j-k} - 1} = 0 \tag{14.6}$$

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 $b := \big(p(1), p(\omega_n^1), \ldots, p(\omega_n^{n-1})\big)$,

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

hat als Input die Länge $n = 2^l$, den n-Vektor b und $\omega = \omega_n$. Das gesuchte Polynom wird auf dem n-Vektor $p = (a_0, \ldots, a_{n-1})$ ausgegeben.

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

Der Algorithmus zur schnellen Multiplikation von Polynomen $p, q \in \mathbb{P}_k$ ergibt sich nun fast von selbst. Wir bestimmen 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. Anschließend berechnen wir $c(k) = a(k)b(k)$ 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.

In der vorgestellten Form ist der Algorithmus aber noch problematisch. Zunächst bringt der Algorithmus nur dann einen beträchtlichen Gewinn gegenüber dem Standardverfahren, wenn $\operatorname{grad} p$ und $\operatorname{grad} q$ von gleicher Größenordnung sind. Beispielsweise hat das Standardverfahren im Extremfall $\operatorname{grad} p = 1$ und $\operatorname{grad} q = n$ die Komplexität $O(n)$, während die schnelle Polynommultiplikation immer noch von der Ordnung $O(n \ln n)$ ist. Dies wird man sicherlich als weniger gravierend ansehen. Bedeutsamer ist dagegen die implizit verwendete Definition des Wortes „Operation“. Auch wenn die Koeffizienten der beteiligten Polynome ganzzahlig sind, werden im Algorithmus nicht rationale komplexe Zahlen benötigt, die auf einem Rechner gar nicht exakt dargestellt werden können. Stattdessen müssen die Operationen mit einer Gleitkommaarithmetik näherungsweise bestimmt werden und es dürfte klar sein, dass man für großes n auch eine umfangreiche Gleitkommaarithmetik benötigt.

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, wobei eine Operation aus der Multiplikation zweier Ziffern oder der Addition zweier ganzer Zahlen besteht. 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. Mit der schnellen Polynommultiplikation kann $p_c(x) = p_a(x)p_b(x)$ bestimmt werden. $p_c(g)$ ist dann das gesuchte Produkt der beiden Zahlen, aus dem noch der Übertrag entfernt werden muss. 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 entsteht.

Da die Rundungsfehleranalyse der diskreten Fouriertransformation aufgrund ihrer rekursiven Struktur recht aufwendig ist, soll hier nur das Endergebnis referiert werden. Wir betrachten den Spezialfall $K = 2^k$ und $g = 2^l$, 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

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

erfüllt ist. Für $k \geq 7$ erhalte wir hieraus die bequemere Abschätzung

$$m \geq 4k + 2l.$$

Um Zahlen der Bitlänge $n = 2^{13} = 8192$ miteinander zu multiplizieren, können wir hier $l = 8$, $k = 11$ setzen und erhalten $m = 54$, was von einer doppelt genauen Gleitkommaarithmetik geleistet wird. Für die meisten praktisch relevanten Fälle lassen sich damit zwei n-stellige Zahlen in $O(n \log n)$ Gleitkommaoperationen miteinander multiplizieren. 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 unitäre 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.