

1 Digitale Signalverarbeitung

1.1 Einführung

Es besteht gar kein Zweifel, wir leben in einer Informationsgesellschaft. Ganz egal, ob wir Musik hören, mit dem Auto unterwegs sind oder mit guten Freunden telefonieren, ständig kommen wir mit Systemen der digitalen Signalverarbeitung in Berührung. Ein großer Teil der Information die uns erreicht ist bereits in mehreren vorangegangenen Schritten digital verarbeitet worden, auch wenn wir uns dessen nicht immer bewußt sind. Träger der Information sind Signale, die im mathematischen Sinne Funktionen einer oder mehrerer unabhängiger Variablen (Zeit, Raumkoordinaten) sind. Die Aufgabe der Signalverarbeitung besteht vor allem in der Manipulation, Analyse und Interpretation von Signalen. Sie kann zu Recht als eine entscheidende Schlüsseltechnologie des modernen Lebens gesehen werden. Interessanterweise wurden die mathematischen Grundlagen hierzu bereits in einer Zeit erarbeitet, als an Computer, Unterhaltungselektronik oder auch nur die Elektrifizierung der privaten Haushalte noch nicht einmal zu denken war.

Einer der Wegbereiter war Jean-Baptiste Joseph Fourier (1768–1830), der im französischen Auxerre als Sohn eines Schneiders geboren wurde. Er besuchte u. A. die dortige Militärschule, wo sein mathematisches Interesse geweckt wurde. Hier machte er durch seine Studien zur Mathematik und Mechanik auf sich aufmerksam. Trotzdem entschied Fourier sich zunächst, nicht unüblich für die damalige Zeit, für das Priesteramt und begann eine entsprechende Ausbildung in der Abtei St. Benoit-sur-Loire. Er mußte sich aber bald eingestehen, daß sein Herz doch mehr für die Mathematik schlug. Zurück in Auxerre arbeitete er als Mathematiklehrer. Die Wirren der Französischen Revolution gingen auch an Fourier nicht spurlos vorüber und hätten ihm beinahe, im wahrsten Sinne des Wortes, Kopf und Kragen gekostet. 1795 zog er nach Paris und vollendete seine Studien an der École Polytechnique, an der er später auch selber lehrte. Fourier war Mitglied der Académie des Sciences und der Académie Française. Sein Ruhm gründet vor allem auf Arbeiten zur Mathematik und zur mathematischen Physik. 1822 entstand sein Hauptwerk, die „Théorie analytique de la chaleur“, in dem er den Wärmetransport und die Temperaturverteilung im



Abb. 1.1. Jean-Baptiste Fourier

Inneren homogener Körper mit Hilfe von partiellen Differentialgleichungen beschreibt.

1.2 Fourier-Reihen

Fourier verwendete in seiner Arbeit trigonometrische Reihen, die man heute als *Fourier-Reihen* kennt. Er erkannte, daß sich periodische Funktionen in einfache Basisfunktionen zerlegen lassen. Die von ihm verwendeten trigonometrischen Funktionen *Sinus* und *Cosinus* bilden gerade einen hierfür geeigneten Funktionsbaukasten. Die notwendige Periodizität liegt genau dann vor, wenn für alle ganzzahligen n die folgende Bedingung erfüllt ist.

$$x(t) = x(t + nT) \quad (1.1)$$

Das bedeutet, der Funktionsverlauf wiederholt sich mit der Periodendauer T , deren Kehrwert bekanntlich die Frequenz ist.

$$f = \frac{1}{T} \quad (1.2)$$

Anstelle der Frequenz f wird jedoch für periodische Vorgänge üblicherweise die sogenannte Kreisfrequenz ω verwendet.

$$\omega = 2\pi f = \frac{2\pi}{T} \quad (1.3)$$

Mit diesem Rüstzeug gelingt die Darstellung einer periodischen Funktion als unendliche Summe der trigonometrischen Basisfunktionen.

$$x(t) = a_0 + \sum_{n=1}^{\infty} (a_n \cos(n\omega t) + b_n \sin(n\omega t)) \quad (1.4)$$

mit

$$a_0 = \frac{1}{T} \int_T x(t) dt \quad (1.5)$$

$$a_n = \frac{2}{T} \int_T x(t) \cos(n\omega t) dt \quad (1.6)$$

$$b_n = \frac{2}{T} \int_T x(t) \sin(n\omega t) dt \quad (1.7)$$

Der Koeffizient a_0 ist der Mittelwert der Zeitfunktion $x(t)$, stellt also den darin enthaltenen Gleichanteil dar. Die Amplituden a_n und b_n repräsentieren die Gewichtung der verschiedenen Frequenzen $n\omega$. Wie später noch gezeigt wird, kommt ihnen bei der Signalanalyse eine ganz besondere Bedeutung zu. In der praktischen Anwendung bricht man eine solche Reihe nach endlich vielen Gliedern ab und begnügt sich mit einer genügend genauen Approximation. Als Beispiel sei an dieser Stelle die Entwicklung einer Rechteckfunktion vorgestellt.

$$\begin{aligned} x(t) &= \sum_{n=1}^{\infty} \frac{\sin((2k-1)\omega t)}{(2k-1)} \\ &= \sin(\omega t) + \frac{\sin(3\omega t)}{3} + \frac{\sin(5\omega t)}{5} + \frac{\sin(7\omega t)}{7} + \dots \end{aligned}$$

In diesem einfachen Beispiel sind keine Cosinus-Anteile enthalten ($a_n=0$ für alle n) und die Amplituden der Sinus-Anteile ergeben sich zu $b_1=1$, $b_2=0$, $b_3=1/3$, $b_4=0$, $b_5=1/5$, $b_6=0$, $b_7=1/7$, ... usw. Die gewünschte Annäherung an den idealen Funktionsverlauf gelingt aber erst mit einer sehr großen Anzahl von Reihengliedern, wie die nachfolgende Abbildung deutlich macht.

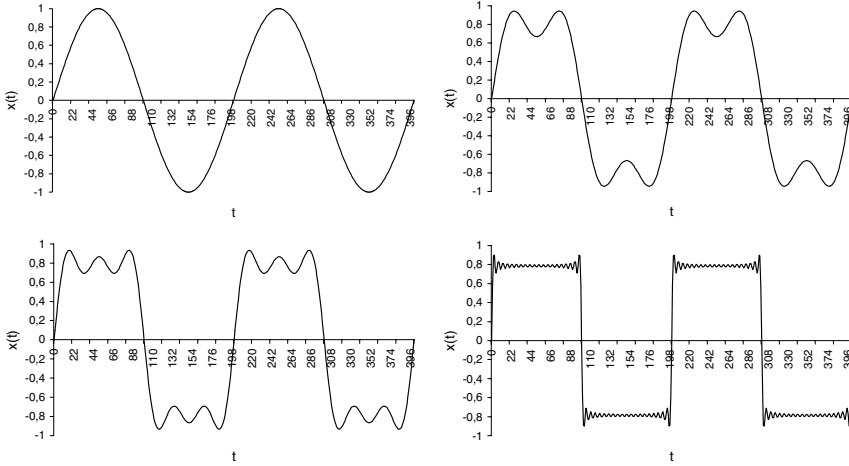


Abb. 1.2. Entwicklung einer Rechteckfunktion mit 1, 2, 3 und 20 Reihengliedern

Offensichtlich bereiten steile Signalflanken gewisse Probleme, denn zu beiden Seiten der Sprungstelle ist ein interessanter Effekt zu beobachten. Es zeigt sich ein deutliches Überspringen, dessen Amplitude auch mit steigender Anzahl von Summanden nicht abnimmt. Dieses Verhalten ist als *Gibbsches Phänomen* bekannt. Die Ursache liegt in der letztlich doch immer nur endlichen Anzahl von Reihengliedern bei einer solchen Approximation. Auch das zweite Beispiel in Tabelle 1.1 zeigt diesen Effekt.

Neben der hier vorgestellten reellen Fourier-Reihe existiert noch eine komplexe Version, die man durch Anwendung der Eulerschen Formel ($e^{j\varphi} = \cos\varphi + j\sin\varphi$) erhält. Die resultierende Reihe kann als orthogonale Entwicklung von $x(t)$ nach einem System komplexer Exponentialfunktionen verstanden werden.

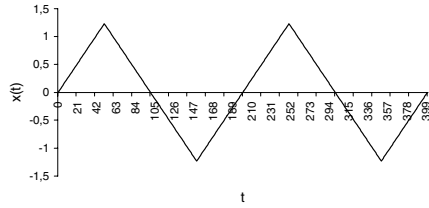
$$x(t) = \sum_{n=-\infty}^{+\infty} c_n e^{jn\omega t} \quad (1.8)$$

$$c_n = \frac{1}{T} \int_T x(t) e^{-jn\omega t} dt \quad (1.9)$$

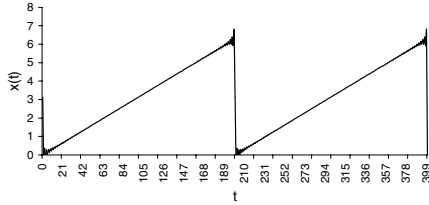
Die von Fourier beschriebene Reihenentwicklung ist nun keineswegs die einzige Möglichkeit eine Funktion $x(t)$ mit Hilfe elementarer Basisfunktionen darzustellen. Vielmehr gibt es beliebig viele solcher Funktionssysteme, die als *Aufbaufunktionen* dienen können, vergleichbar mit einem Vektor, der ebenfalls in verschiedenen Koordinatensystemen (rechtwinklig, schiefwinklig, Polarkoordinaten, ...) darstellbar ist. Allerdings kommt

Tabelle 1.1. Beispiele für Fourier-Reihen

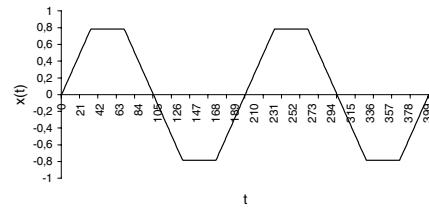
$$x(t) = \sum_{n=1}^{\infty} (-1)^{n+1} \cdot \frac{\sin((2n-1)\omega t)}{(2n-1)^2}$$



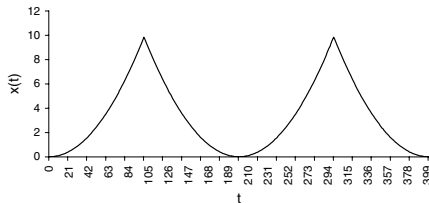
$$x(t) = \pi - 2 \left(\sum_{n=1}^{\infty} \frac{\sin(n\omega t)}{n} \right)$$



$$x(t) = \sum_{n=1}^{\infty} \frac{\sin(2n-1) \cdot \sin((2n-1)\omega t)}{(2n-1)^2}$$



$$x(t) = \frac{\pi^2}{3} - 4 \left(\sum_{n=1}^{\infty} (-1)^{n+1} \cdot \frac{\cos(n\omega t)}{n^2} \right)$$



es in einer Anwendung ganz entschieden darauf an, die jeweils günstigste Darstellung zu wählen. Das gilt auch für die Wahl des Funktionssystems bei der Approximation von $x(t)$. Fourier verwendete für seine periodischen Funktionen die trigonometrischen Basisfunktionen Sinus und Cosinus, und er tat gut daran. Wie sich wahrscheinlich jeder leicht vorstellen kann, ist es immer günstig gerade solche Aufbaufunktionen zu wählen, die eine gewisse Ähnlichkeit mit $x(t)$ aufweisen, da die Approximation dann besonders schnell konvergiert. Eine derartige Reihenentwicklung läßt sich ganz allgemein formulieren:

$$x(t) = \sum_{n=0}^{\infty} a_n \varphi_n(t) \quad (1.10)$$

Wie auch schon bei der Fourier-Reihe, ist $x(t)$ eine Linearkombination der Aufbaufunktionen $\varphi_n(t)$, wobei es gilt, die Entwicklungskoeffizienten a_n zu bestimmen. Hierzu sind beide Seiten der Gleichung mit den konjugiert komplexen Funktionen $\varphi_k^*(t)$ zu multiplizieren und über das Intervall T zu integrieren. Der Rechenaufwand vereinfacht sich erheblich, wenn das gewählte Funktionssystem $\varphi_n(t)$ die *Orthogonalitätsbedingung* erfüllt.

$$\int_T \varphi_n(t) \cdot \varphi_m^*(t) dt = \begin{cases} 0 & (n \neq m) \\ k_n & (n = m) \end{cases} \quad (1.11)$$

Sind zusätzlich alle $k_n = 1$, ist $\varphi_n(t)$ als orthonormal. Die Multiplikation von Gln. (1.10) mit $\varphi_k^*(t)$ und anschließende Integration führen auf die nachfolgende Beziehung.

$$\int_T x(t) \cdot \varphi_m^*(t) dt = \int_T \sum_{n=0}^{\infty} a_n \varphi_n(t) \cdot \varphi_m^*(t) dt$$

Vorausgesetzt die Summe (1.10) ist gleichmäßig konvergent und man beachtet die Bedingung (1.11), folgt hieraus die Formel zur Berechnung der gesuchten Koeffizienten.

$$a_n = \frac{1}{k_n} \int_T x(t) \cdot \varphi_n^*(t) dt \quad (1.12)$$

Neben den trigonometrischen Aufbaufunktionen sind noch die 1923 von J. L. Walsh definierten *Walsh-Funktionen* und die nur ein Jahr zuvor von H. Rademacher entwickelten *Rademacher-Funktionen* von praktischer Bedeutung. Beide sind Erweiterungen der 1910 von A. Haar vorgestellten orthogonalen Funktionsbasis, den sogenannten *Haar-Funktionen*.

1.3 Die Diskrete Fourier-Transformation (DFT)

Schaut man auf das bisher Gezeigte, liegt die Vermutung nahe, daß es auch eine Möglichkeit gibt, den umgekehrten Weg zu beschreiten, also eine Funktionsapproximation wieder in ihre erzeugenden Basisfunktionen zu zerlegen. Ein solches Vorgehen wäre dann geeignet, ein Signal in Bezug auf seine spektrale Zusammensetzung zu analysieren. Während man die Gleichungen (1.4) und (1.8) häufig auch als *Synthesegleichungen* bezeichnet, beschreiben die Gleichungen (1.5), (1.6), (1.7) und (1.9) das Signal im Frequenzbereich und werden daher entsprechend *Analysegleichungen* genannt. Ein Digitalrechner kann aber immer nur zeitdiskrete Werte erfassen und verarbeiten. Die zeitkontinuierlichen Signalverläufe realer

physikalischer Größen müssen infolgedessen diskretisiert werden. Möchte man eine Frequenzanalyse für ein Abtastsignal durchführen, müssen dann auch die Transformationen (1.6) und (1.7) zeitdiskret angepaßt werden. Dabei gehen die zeitkontinuierlichen Integrale der orthogonalen Komponenten, die den Realteil und den Imaginärteil des Signals repräsentieren, in zeitdiskrete Summen über.

$$a_n = \frac{2}{N} \sum_{k=0}^{N-1} x(k) \cdot \cos(2\pi n \frac{k}{N}) \quad (1.13)$$

$$b_n = \frac{2}{N} \sum_{k=0}^{N-1} x(k) \cdot \sin(2\pi n \frac{k}{N}) \quad (1.14)$$

$$|X_n| = \sqrt{a_n^2 + b_n^2} \quad (1.15)$$

$$\varphi_n = \arctan\left(\frac{b_n}{a_n}\right) \quad (1.16)$$

Aus diesen Summen können dann wiederum nach Gln. (1.15) das *Amplitudenspektrum*, oft auch *Betragspektrum* genannt, und das *Phasenspektrum* nach Gln. (1.16) bestimmt werden. Hierbei stellt k die diskretisierten Abtastzeitpunkte dar, N die Anzahl der Abtastwerte je Periode der Grundfrequenz und n die gerade betrachtete Frequenz. Sie sind somit die diskreten Gegenstücke zu den zeitkontinuierlichen Größen t , T und ω . Insgesamt erfordert die Frequenzanalyse eines abgetasteten Signals mit Hilfe der vorgestellten reellen *DFT* nur einen vergleichsweise geringen Implementierungsaufwand und eignet sich damit ganz hervorragend für eigene experimentelle Untersuchungen.

```
class Spektrum
{
    const int Abtastwerte = 128;
    const double PI      = 3.141592653589793;

    // Signalverlauf
    public double[] x  = new double[Abtastwerte];
    // Realteil und Imaginärteil
    public double[] Re = new double[Abtastwerte];
    public double[] Im = new double[Abtastwerte];
    // Amplitudenspektrum und Phasenspektrum
    public double[] AS = new double[Abtastwerte];
    public double[] PS = new double[Abtastwerte];
}
```

```

...
public void DFT()
{
    // über alle Frequenzen
    for (int f=0; f<Abtastwerte; f++)
    {
        Re[f] = 0;
        Im[f] = 0;

        // über alle Abtastwerte
        for (int k=0; k<Abtastwerte; k++)
        {
            Re[f] += x[k]*Math.Cos(2*PI*f*k/Abtastwerte);
            Im[f] += x[k]*Math.Sin(2*PI*f*k/Abtastwerte);
        }
        Re[f] *= 2.0/Abtastwerte;
        Im[f] *= 2.0/Abtastwerte;

        AS[f] = Math.Sqrt(Re[f]*Re[f] + Im[f]*Im[f]);
        PS[f] = Math.Atan2(Im[f], Re[f]);
    }
    Re[0] /= 2.0; // Korrektur Gleichanteil
    AS[0] = Re[0];
}
...
}

```

Die Korrektur des Gleichanteils wird vorgenommen, da in der Regel a_0 nicht nach Gln. (1.5) berechnet wird, sondern nach Gln. (1.13). Man kann bei der Implementierung aber auch auf die Korrektur verzichten. In der Praxis kommt das sogar relativ häufig vor, wohlwissend, daß die Spektrallinie AS[0] dann dem doppelten Gleichanteil des Signals entspricht.

In den meisten Fällen ist es völlig ausreichend, sich bei der Darstellung des Signalspektrums auf das Amplitudenspektrum zu beschränken. Es ist für jeden Signalabschnitt identisch und dadurch von eventuellen zeitlichen Verschiebungen des periodischen Signals unabhängig. Ganz anders sieht das aber beim Phasenspektrum aus, das sich aus dem Quotienten von Imaginärteil und Realteil berechnet. Wie man leicht nachvollziehen kann, ist das Phasenspektrum sehr wohl abhängig von dem gewählten zeitlichen Bezugspunkt. Aus diesem Grund ist es sehr viel weniger aussagekräftig und trägt nicht selten sogar eher zur Verwirrung bei. In den folgenden Beispielen betrachten wir daher ausschließlich das Amplitudenspektrum.

Sollte es die Aufgabenstellung erfordern den zeitlichen Verlauf eines Signals aus einem vorgegebenen Spektrum zu bestimmen, ist das ebenfalls

leicht möglich und vollzieht sich in zwei Schritten. Aus dem Amplitudenspektrum ist zwar sofort erkennbar, wie stark die einzelnen Frequenzen vertreten sind, es ist aber zunächst nicht klar, ob diese als Sinus- oder als Cosinusanteil in den Signalverlauf eingehen. Wie schon bei der Signalanalyse, sind daher durch Korrelation mit den trigonometrischen Aufbaufunktionen der Realteil und der Imaginärteil des Spektrums zu ermitteln. Mit dessen orthogonalen Komponenten als Gewichtungsfaktoren gelingt dann die Synthese des zeitlichen Verlaufs. Eine erste Implementierungsvariante der inversen DFT (*IDFT*) zeigt das folgende Code-Beispiel.

```
class Spektrum
{
    ...
    public void InverseDFT_Variante1()
    {
        // über alle Frequenzen
        for (int f=0; f<Abtastwerte; f++)
        {
            // Berechnung der orthogonalen Komponenten von
            // XS(f) durch Korrelation mit Cos und Sin
            Re[f] += XS[f]*Math.Cos(2*PI*f/Abtastwerte);
            Im[f] += XS[f]*Math.Sin(2*PI*f/Abtastwerte);

            // Synthese von x[k] durch Gewichtung
            // der orthogonalen Komponenten von XS(f)
            // mit den Aufbaufunktionen
            for (int k=0; k<Abtastwerte; k++)
            {
                x[k] += Re[f]*Math.Cos(2*PI*f*k/Abtastwerte)+
                    Im[f]*Math.Sin(2*PI*f*k/Abtastwerte);
            }
        }
    }
    ...
}
```

Wie schon bei der DFT sind auch bei der IDFT zwei geschachtelte Schleifen für die Transformation notwendig. Die Berechnung der orthogonalen Spektralkomponenten erfolgt zunächst in der äußeren Schleife, die Synthese des zeitlichen Verlaufs läuft dann in der inneren Schleife ab. Ähnlich wie bei einer mathematischen Formel kann man aber auch zweckmäßig umformen. In unserer zweiten Implementierung vereinfacht sich dadurch die Synthese zur simplen Addition.

```
class Spektrum
{
    ...
    public void InverseDFT_Variante2()
    {
        for (int k=0; k<Abtastwerte; k++)
        {
            Re[k] = 0;
            Im[k] = 0;

            for (int f=0; f<Abtastwerte; f++)
            {
                Re[k] += XS[f]*Math.Cos(2*PI*k*f/Abtastwerte);
                Im[k] += XS[f]*Math.Sin(2*PI*k*f/Abtastwerte);
            }

            x[k] = Re[k] + Im[k];
        }
    }
    ...
}
```

Diese zweite Variante der IDFT ähnelt der DFT doch sehr auffällig. Das ist natürlich kein Zufall, sondern liegt in der schlichten Tatsache begründet, daß die Hin- bzw. Rücktransformation einander sehr ähnlich sind. Allgemein läßt sich das an den Gleichungen (1.10) und (1.12) erkennen, speziell für die Fourier-Transformation an den Gleichungen (1.8) und (1.9). Diskretisiert man hier die Analysegleichung indem man das Integral durch die Summe ersetzt, unterscheiden sich Hin- bzw. Rücktransformation nur durch eine Normierung und verschiedene Vorzeichen in der komplexen Exponentialfunktion. Eine komplexe Exponentialfunktion ist aber nichts weiter als ein Drehzeiger in der komplexen Zahlenebene, wobei das Vorzeichen die Drehrichtung angibt. Ein Wechsel des Vorzeichen ist hier etwa vergleichbar mit der Summation von links nach rechts, oder umgekehrt. Das Ergebnis ist in jedem Fall genau dasselbe. Somit sind auch die Implementierung der DFT und der IDFT im Kern identisch. In Anwendungen, in denen tatsächlich auch beide Transformationen benötigt werden, sind diese dann üblicherweise in einer einzigen Funktion realisiert. Ein zusätzlicher Parameter beim Funktionsaufruf dient zur Unterscheidung. Aus optischen Gründen ist es dann zweckmäßig anstelle von `x[]` und `XS[]` neutrale Bezeichner wie `data1[]` und `data2[]` zu verwenden. In keinem Fall darf man jedoch vergessen, daß die Ergebnisse im Frequenz- bzw. Zeitbereich entsprechend unterschiedlich skaliert sind.

1.4 Arithmetiktuning – Die FFT

Das Frequenzspektrum einer Folge von Abtastwerten zu berechnen, ist jetzt kein Problem mehr. Mit Hilfe der DFT können wir auf die erweiterten Möglichkeiten einer Meßwertanalyse im Frequenzbereich zurückgreifen. Doch wieviel Rechenzeit darf eine solche Analyse in Anspruch nehmen? Oft ist die DFT nur ein Glied in einer ganzen Kette von Verarbeitungsschritten. Bei zeitkritischen Anwendungen, wie z. B. der Sprachsignalverarbeitung, kommt es dann schon darauf an, den Rechenzeitbedarf insgesamt möglichst gering zu halten. Erfahrungsgemäß ist das Einsparpotential gerade dort am größten, wo auch die meiste Rechenzeit verbraucht wird.

Tatsächlich ist der Rechenzeitbedarf in vielen Anwendungen der digitalen Signalverarbeitung ein latentes Problem, dem man grundsätzlich auf zwei Arten begegnen kann. Einerseits durch Optimierung der verwendeten Algorithmen, als Softwareentwickler können wir hier unmittelbar Einfluß nehmen, andererseits durch die Verwendung einer entsprechend leistungsstarken Hardwarebasis, wobei jedoch der Kostenrahmen des Projektes meist enge Grenzen setzt. Es liegt in der Natur der Sache, daß beide Ansätze das Problem immer nur kurzzeitig entschärfen, aber nicht wirklich dauerhaft lösen können, da sich mit den dann zusätzlichen Möglichkeiten auch immer komplexere Aufgabenstellungen finden, die diesen Freiraum schnell wieder aufzehren. Somit bleibt die Rechenzeitproblematik mit großer Sicherheit auch zukünftig bestehen und jeder Entwickler ist gut beraten, dies in seinem Projekt möglichst frühzeitig zu berücksichtigen.

Viele Wege führen nach Rom. Und wahrscheinlich gibt es genauso viele Berechnungsvarianten für die DFT einer Abtastfolge. Eine Variante, die sogenannte Fast-Fourier-Transformation (*FFT*), hat sich jedoch einen ganz besonderen Stellenwert innerhalb der digitalen Signalverarbeitung erobert. Sie gehört ohne Zweifel zur „Königsklasse“ der Signalverarbeitungsalgorithmen und ist wohl auch einer ihrer kompliziertesten Vertreter. Hiervon sollte man sich aber nicht weiter beeindrucken lassen. Im Grunde ist alles doch ganz einfach.

Wie wir zuvor gesehen haben, sind für die Berechnung einer DFT mit N Abtastwerten $N \cdot N = N^2$ komplexe Multiplikationen notwendig. Man kann sich leicht vorstellen, daß eine solche quadratische Abhängigkeit in Anwendungen mit großen Abtastraten Probleme schafft. In solchen Fällen hat sich das Prinzip *Teile-und-Herrsche* (divide-and-conquer) bewährt. Eine aufwendige Berechnung wird in viele kleine Teilberechnungen zerlegt und deren Teilergebnisse anschließend zu einem Gesamtergebnis zusammengeführt. Das sich dieses Vorgehen auch für eine schnellere Berechnung der DFT eignet, wurde schon frühzeitig erkannt, lange bevor es überhaupt

Computer gab. Aber erst die elektronische Rechentechnik hat dieser Idee zum Durchbruch verholfen. Heute werden insbesondere die Arbeiten von Danielson und Lanczos aus dem Jahre 1942, sowie von Cooley und Tukey aus dem Jahre 1965 als Wiederentdeckung der FFT gesehen.

Um das Teile-und-Herrsche-Prinzip zu verdeutlichen betrachten wir zunächst die diskrete Fouriertransformation. Diese läßt sich durch Zerlegung in gerade und ungerade Abtastwerte wie folgt umschreiben:

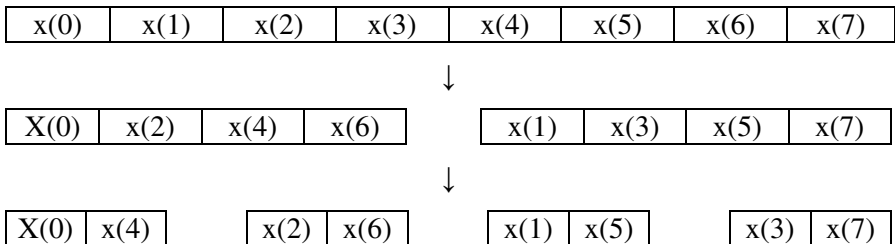
$$X_n = \sum_{k=0}^{N-1} x(k) \cdot e^{-\frac{j2\pi nk}{N}}$$

$$X_n = \sum_{k=0}^{N/2-1} x(2k) \cdot e^{-\frac{j2\pi n(2k)}{N}} + \sum_{k=0}^{N/2-1} x(2k+1) \cdot e^{-\frac{j2\pi n(2k+1)}{N}}$$

$$X_n = \sum_{k=0}^{N/2-1} x(2k) \cdot e^{-\frac{j2\pi nk}{N/2}} + \left(\sum_{k=0}^{N/2-1} x(2k+1) \cdot e^{-\frac{j2\pi nk}{N/2}} \right) \cdot e^{-\frac{j2\pi n}{N}}$$

Eine DFT der Länge N kann also auch aus zwei DFTs der halben Länge berechnet werden, wenn man sie anschließend phasenrichtig addiert. Was das bringt liegt auf der Hand. Anstelle einer N^2 Berechnung hat man es nur noch mit $2 \cdot (N/2) \cdot (N/2) = N^2/2$ Multiplikationen zu tun, was den Aufwand glatt halbiert. Aber es kommt noch besser. Dieses Vorgehen läßt sich rekursiv fortführen, solange die verbleibenden Teilintervalle durch Zwei teilbar sind. Theoretisch könnte man so weitermachen, bis man N DFTs aus N Einzelwerten zu berechnen hat. Das Ergebnis wäre dann der jeweilige Wert selbst. In der Praxis hat es sich aber durchgesetzt zwei Abtastwerte für eine Minimal-DFT zu verwenden. So eine Minimal-DFT wird dann als *Butterfly* bezeichnet.

Natürlich ändert sich durch die fortlaufende Zerlegung in gerade und ungerade Teilfolgen auch die Reihenfolge unserer ursprünglich zeitlich geordneten Abtastwerte. Die Indizes sind nicht mehr aufsteigend sortiert, sondern werden nach einem neuen Schema umgruppiert, wie das nachstehende Beispiel mit $N = 8$ Abtastwerten zeigt.



Der erste Schritt zerlegt die acht Abtastwerte in zwei Teilfolgen mit jeweils vier Werten. In einem zweiten Schritt werden diese nun ihrerseits zerlegt und wir erhalten insgesamt vier Teilfolgen, bestehend aus je zwei Werten. Dieser Vorgang ist an sich nicht weiter kompliziert. Sehr viel schwieriger ist es allerdings die Gesetzmäßigkeit hinter der resultierenden Neuordnung zu erkennen, zumindest solange wir die Indizes ausschließlich als Dezimalzahlen betrachten. Im Dualsystem dagegen offenbart sich das Muster sehr schnell. Der Index 1 (binär 001) wird mit dem Index 4 (binär 100) getauscht, der Index 3 (binär 011) mit der 6 (binär 110), usw., was gerade einer *Bit-Umkehr* (bit reversal) entspricht.

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

Mit Hilfe der Bit-Umkehr ist es also möglich, die oben beschriebene Zerlegung in gerade und ungerade Teilfolgen nicht explizit durchführen zu müssen, aber dennoch auf identisch angeordnete Abtastwerte zugreifen zu können.

Wir erinnern uns, der ursprüngliche Sinn der Zerlegung war es, mehrere Teil-DFTs zu berechnen und diese dann zum richtigen Gesamtergebnis zusammenzuführen, um Aufwand zu sparen. Die Zusammenführung ist aber gerade die Umkehrung der oben vorgenommenen Zerlegung. Hierbei sind zuerst vier 2-Punkte DFTs aus den Paaren $x(0)$ und $x(4)$, $x(2)$ und $x(6)$, $x(1)$ und $x(5)$ und schließlich $x(3)$ und $x(7)$ zu bilden. Als Ergebnis erhalten wir vier 2-Punkte Spektren, die nun ihrerseits als Eingangswerte für zwei 4-Punkte DFTs dienen. Die beiden resultierenden 4-Punkte Spektren werden abschließend noch einer 8-Punkte DFT unterzogen, und schon haben wir unser Endergebnis. Zugegeben, die Interpretation der Zwischenergebnisse als Spektren ist ein wenig problematisch, schaut man aber auf den Berechnungsaufwand, dann ist das Ergebnis beeindruckend. Für unsere $N = 8$ Abtastwerte sind demnach nur noch $4 \cdot 2 + 2 \cdot 4 + 1 \cdot 8 = 24$ komplexe Multiplikationen nötig, was doch ein ganz ordentliches Stück besser ist als die 64 bei der DFT. Allgemein sind mit dem hier vorgestellten FFT-Algorithmus $N \cdot \log_2(N)$ Schritte erforderlich, gegenüber N^2 Schritte bei der normalen DFT.

Neben der Zerlegung bzw. Neuordnung der Abtastwerte ist die phasenrichtige Addition bei den Teil-DFTs die entscheidende Hürde zum Erfolg.

Wir wollen diese daher näher betrachten. In unserer nach geraden und ungeraden Abtastwerten zerlegten Beispielberechnung der DFT taucht ein zusätzlicher Faktor auf, der, abgesehen von der aktuellen Frequenz f , nur von der Anzahl der Abtastwerte N abhängt. Eine solche komplexe Exponentialfunktion bildet einen *Drehzeiger* (twiddle faktor) in der komplexen Zahlenebene. Genau dieser Zeiger ist das fehlende Glied, das für die Phasenkorrektur unserer Teil-DFT verantwortlich ist.

$$w_N = e^{-\frac{j2\pi f}{N}} \quad (1.17)$$

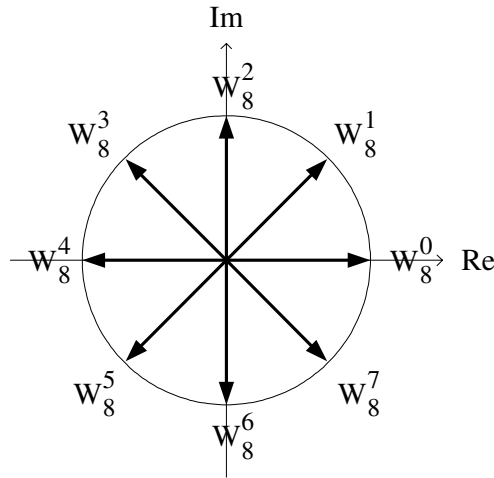


Abb. 1.3. Drehzeiger für $N = 8$

Die Zeiger liegen symmetrisch auf dem Einheitskreis ($r = 1$) und bilden die N Einheitswurzeln. Für die Implementierung wird es etwas einfacher, wenn man Gln. (1.17) unter Verwendung der trigonometrischen Funktionen umschreibt, wobei der Index k alle Zeiger von 0 bis $N-1$ durchläuft:

$$w_N^k = \cos\left(\frac{2\pi k}{N}\right) + j \sin\left(\frac{2\pi k}{N}\right) \quad (1.18)$$

In dieser Form lassen sich Real- und Imaginärteil des Drehzeigers direkt berechnen. Nutzt man seine Symmetrieeigenschaft, kann man zusätzlich Berechnungsaufwand sparen.

$$w_8^4 = -w_8^0$$

$$w_8^5 = -w_8^1$$

$$w_8^6 = -w_8^2$$

$$w_8^7 = -w_8^3$$

Die Neuordnung der Abtastwerte und der eingeführte Drehzeiger ermöglichen schließlich das effektive Berechnungsschema der FFT.

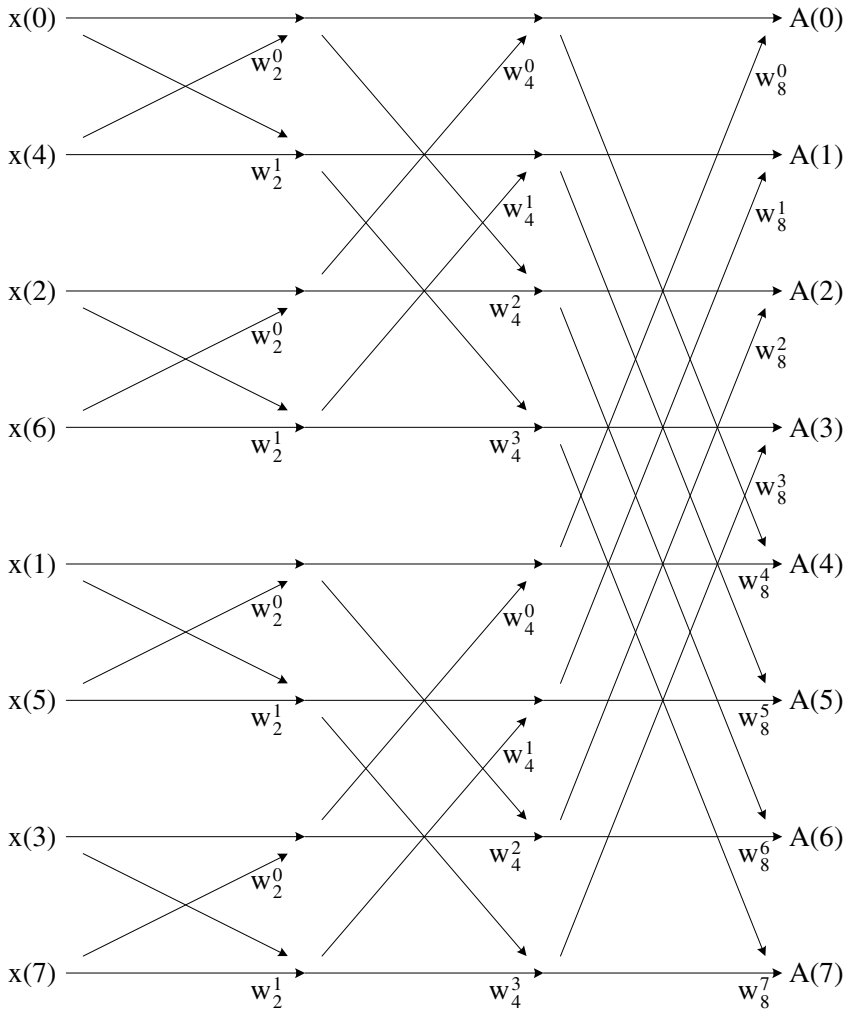


Abb. 1.4. Berechnungsschema für eine FFT mit 8 Abtastwerten

Bei acht Abtastwerten vollzieht sich die Berechnung in drei Stufen. Die Zusammenführung der Pfeile stellt eine Addition dar, ein beistehender Drehzeiger bedeutet eine Multiplikation mit dem entsprechenden Wert. Verständlicher wird es, wenn man einen einzelnen Butterfly betrachtet.

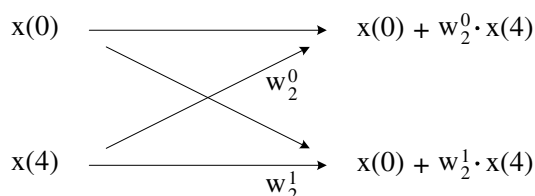


Abb. 1.5. Berechnungsschema eines einzelnen Butterfly

Für das Arbeiten mit komplexen Zahlen benötigt man noch einen entsprechenden Datentyp. Die Berechnungen selbst erfolgen dann jeweils getrennt nach Realteil und Imaginärteil.

```
struct komplex
{
    public double re;
    public double im;
}
```

Echte Meßwerte aus realen Prozessen sind immer reellwertig. In der Regel werden sie auch als solche gespeichert und weiterverarbeitet (üblicherweise noch mit Zeitstempel, physikalischer Einheit, usw.). Für die FFT ist daher die vorherige Umwandlung in komplexe Daten erforderlich.

Auch wenn man bis zu diesem Punkt alles verstanden hat, die wohl größte Schwierigkeit bei der Implementierung liegt ohne Zweifel darin, das Berechnungsschema der FFT über eine geeignete Indizierung tatsächlich umzusetzen. Hierzu führen wir drei Hilfsvariablen ein. Die Variable *stufen* bezeichnet die Anzahl der Berechnungsebenen im Schema und ergibt sich aus dem dualen Logarithmus der Zahl der Abtastwerte ($8=2^3 \rightarrow$ drei Stufen). Mit *sprung* wird die notwendige Inkrementierung des Indexes bezeichnet, um vom ersten Zweig des Butterflys zum zweiten Zweig zu gelangen ($=4$ in der ersten Ebene: $0 \rightarrow 4$, $2 \rightarrow 6$, $1 \rightarrow 5$ und $3 \rightarrow 7$). Die Variable *schritt* stellt das notwendige Inkrement dar, um vom ersten Zweig eines Butterflys zum ersten Zweig des nächsten Butterflys zu springen ($=2$ in der ersten Ebene: $0 \rightarrow 2$ und $1 \rightarrow 3$).

```

class Spektrum
{
    ...
    // komplexe Daten
    public komplex[] data = new komplex[Abtastwerte];

    public void FFT()
    {
        double tempr = 0; // für Tausch bei Bit-Umkehr
        double tempi = 0; // für Tausch bei Bit-Umkehr
        double wreal = 0; // Drehfaktor (Realteil)
        double wimag = 0; // Drehfaktor (Imaginärteil)
        double real1 = 0; // Hilfsvariable
        double imag1 = 0; // Hilfsvariable
        double real2 = 0; // Hilfsvariable
        double imag2 = 0; // Hilfsvariable
        int i = 0;
        int j = 0;
        int k = 0;
        int stufen = 0;
        int sprung = 0;
        int schritt = 0;
        int element = 0;

        // Bit-Umkehr
        for (j=0; j<Abtastwerte-1; j++)
        {
            if (j < i)
            {
                tempr = data[j].re;
                tempi = data[j].im;
                data[j].re = data[i].re;
                data[j].im = data[i].im;
                data[i].re = tempr;
                data[i].im = tempi;
            }

            k = Abtastwerte/2;
            while (k <= i)
            {
                i -= k;
                k /= 2;
            }

            i += k;
        }
    }
}

```

```
stufen = (int) (Math.Log10((double)Abtastwerte) /  
               Math.Log10((double)2));  
sprung = 2;  
  
for (i=0; i<stufen; i++)  
{  
    // jede Iterationsstufe startet mit dem 1. Wert  
    element = 0;  
  
    for (j=Abtastwerte/sprung; j>=1; j--)  
    {  
        schritt = sprung/2;  
  
        for (k=0; k<schritt; k++)  
        {  
            // 1. Zweig des Butterfly  
            wreal = Math.Cos(k*2.0*PI/sprung);  
            wimag = Math.Sin(k*2.0*PI/sprung);  
  
            real1 = data[element+k].re +  
                    (wreal*data[element+k+schritt].re -  
                     wimag*data[element+k+schritt].im);  
            imag1 = data[element+k].im +  
                    (wreal*data[element+k+schritt].im +  
                     wimag*data[element+k+schritt].re);  
  
            // 2. Zweig des Butterfly  
            wreal *= -1.0;  
            wimag *= -1.0;  
  
            real2 = data[element+k].re +  
                    (wreal*data[element+k+schritt].re -  
                     wimag*data[element+k+schritt].im);  
            imag2 = data[element+k].im +  
                    (wreal*data[element+k+schritt].im +  
                     wimag*data[element+k+schritt].re);  
  
            // Ergebnisse übernehmen  
            data[element+k].re = real1;  
            data[element+k].im = imag1;  
            data[element+k+schritt].re = real2;  
            data[element+k+schritt].im = imag2;  
        }  
        element += sprung;  
    }  
    sprung *= 2;  
}
```

```

    }
}
...
}

```

Wie schon bei der DFT unterscheidet sich die inverse FFT (IFFT) von der FFT lediglich durch eine andere Normierung, bzw. Skalierung des Ergebnisses. Unser ursprüngliches Anliegen war es, Rechenzeit zu sparen. Geht man allein nach der Anzahl der benötigten Programmzeilen, dann scheinen wir von diesem Ziel weiter entfernt als zuvor. Aber wer wird sich schon von Äußerlichkeiten blenden lassen, was zählt sind schließlich die inneren Werte. Führt man sich vor Augen, wie viele komplexe Multiplikationen bei DFT bzw. FFT auszuführen sind, wird schnell klar, welches Verfahren die Nase vorn hat.

Ob sich die Einsparung an Rechenoperationen bei der FFT auch im gleichen Maße im Rechenzeitbedarf niederschlägt, läßt sich gut mit Hilfe eines Benchmark-Tests überprüfen. Hierbei wird die Ausführungszeit der Algorithmen miteinander verglichen. Um den relativen Fehler klein zu halten, wird pro Zeitmessung immer eine größere Anzahl von Durchläufen (z. B. 1000) ausgeführt. Wegen der Verwendung eines eigenen Datentyps, wurde bei der FFT die Zuweisung der Meßwerte mit in die Testschleife einbezogen. Genau genommen müßte man noch deren Leerlaufzeit abziehen, diese ist hier aber vernachlässigbar.

Tabelle 1.2. Anzahl der komplexen Multiplikationen bei DFT und FFT

N	DFT	FFT
2	4	2
4	16	8
8	64	24
16	256	64
32	1024	160
64	4096	384
128	16384	896
256	65536	2048
512	262144	4608
1024	1048576	10240

```

class Spektrum
{
    ...
    public void Benchmark()
    {

```

```
const int testzyklen = 1000;

DateTime start;
DateTime stop;

Console.WriteLine("Benchmark DFT startet jetzt!");
start = DateTime.Now;

for (int j=0; j<testzyklen; j++)
{
    DFT();
}

stop = DateTime.Now;

Console.WriteLine("Dauer = {0}", stop-start);
Console.WriteLine();

Console.WriteLine("Benchmark FFT startet jetzt!");
start = DateTime.Now;

for (int j=0; j<testzyklen; j++)
{
    for (int k=0; k<Abtastwerte; k++)
    {
        data[k].re = x[k];
        data[k].im = 0;
    }
    FFT();
}

stop = DateTime.Now;

Console.WriteLine("Dauer = {0}", stop-start);
Console.ReadLine();
}
...
}
```

Natürlich sind die gemessenen Zeiten in erster Linie von der Leistungsfähigkeit des ausführenden Computers abhängig. Eine normierte Darstellung der Werte ist in solchen Fällen hilfreich.

Tabelle 1.3. Gemessener Zeitbedarf von DFT und FFT (normiert)

N	DFT	FFT
2	1,3	1
4	5,7	2,6
8	24	7
16	94	18
32	355	44
64	1425	103
128	5703	242
256	22972	528
512	92551	1111
1024	371194	2778

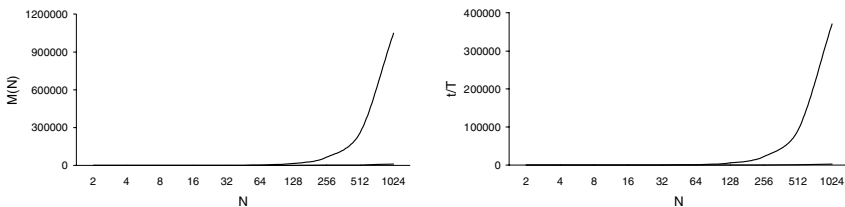
**Abb. 1.6.** Vergleich theoretischer und gemessener Zeitbedarf von DFT und FFT

Abbildung 1.6 zeigt die gute Übereinstimmung von Theorie und Praxis beim Rechenzeitbedarf beider Verfahren. Die Kurve für die FFT schmiegt sich so dicht an die x -Achse an, daß sie kaum sichtbar wird. Für kleine Abtastraten ist der Unterschied weit weniger dramatisch. Tatsächlich gibt es dort für die DFT noch erhebliches Einsparpotential, z. B. indem man die zeitaufwendigen trigonometrischen Funktionsaufrufe für alle k direkt durch ihren jeweiligen Wert ersetzt. Für höhere Abtastraten ist das aber kaum praktikabel, so daß sich die FFT etwa für $N > 16$ lohnt.

Vom Ansatz her produzieren DFT und FFT identische Ergebnisse. Die DFT berechnet die Frequenztransformierte einer diskreten Abtastfolge und genau das gleiche macht auch die FFT. Eigentlich ist sie ja auch nur eine (deutlich effizientere) Berechnungsvariante der DFT. Hierbei könnte man es belassen, wenn da nicht die Zahlendarstellung im Computer mit ihrem begrenzten Auflösungsvermögen wäre. Da es nicht möglich ist alle reellen Zahlenwerte in einem Rechner exakt abzubilden, geht jeder Berechnungsschritt mit einem gewissen Verlust an Präzision einher. Das Ergebnis der FFT ist aus diesem Grund zumindest theoretisch näher am „wahren Wert“, da sich mit der geringeren Anzahl von Berechnungen auch weniger Rundungsfehler aufsummieren. In der Praxis dürfte sich dieser Unterschied allerdings kaum bemerkbar machen. Wer es genauer wissen möchte, kann

das durch eine große Anzahl von hintereinander ausgeführten Hin- und Rücktransformationen selbst austesten (ähnlich dem Qualitätstest von Fotokopierern, wo man eine Kopie von einer Kopie von einer Kopie...).

1.5 Pulse und Pulsfolgen

Um das bisher Erarbeitete auch praktisch anwenden zu können, bedarf es natürlich noch einiger Testsignale. Da echte Signale aus realen technischen Prozessen leider nicht immer zur Verfügung stehen, liegt es nahe, eigene Signalfolgen zu erstellen und diese synthetischen Daten als Basis für die weiteren Untersuchungen zu verwenden. Dabei ist es zunächst einmal egal, ob tatsächlich eine periodische Funktion vorliegt, oder ob ein Signal betrachtet wird, von dem wir lediglich annehmen, daß es periodisch ist. Natürlich könnte man einfach unter Verwendung von Gln. (1.4) ein Signal nach einer Fourier-Reihe entwickeln und anschließend mittels DFT untersuchen. Als Ergebnis bekäme man dann jedoch nur genau das heraus, was man selbst zuvor hineingesteckt hat. Von einem großen Erkenntnisgewinn kann da nun wirklich nicht die Rede sein. In der Signalverarbeitung bilden Pulse und Pulsfolgen eine besondere Klasse von Signalen, weshalb es sich lohnt, sich näher mit ihnen zu befassen. Sie haben zudem den großen Vorteil, daß sie sich softwaretechnisch sehr leicht erzeugen lassen, einige wenige Zeilen genügen.

```
class Spektrum
{
    ...
    public void ErzeugeTestsignal_1()
    {
        for (int k=0; k<Abtastwerte; k++)
        {
            if (k < Abtastwerte/16)
                x[k] = 1;
            else
                x[k] = 0;
        }
    }
    ...
}
```

Als Ergebnis erhält man den oben dargestellten Rechteckimpuls von 1/16-Periodendauer Länge. Schaut man sich anschließend das berechnete

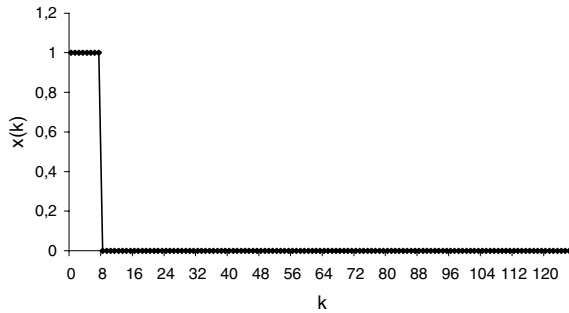


Abb. 1.7. Rechteckimpuls von $1/16$ Länge

Amplitudenspektrum an, ist man vielleicht überrascht, welche Frequenzkomponenten schon in so einem einfachen Impuls enthalten sind. Im Allgemeinen sind deren Anteile allein durch Betrachtung des zeitlichen Signalverlaufs kaum zu erraten. Eine Ausnahme bildet hier der Gleichanteil. Weiterhin fällt sofort die Spiegelung des Spektrums an der Stelle $N/2$ auf. Offensichtlich genügt es für eine reelle DFT nur $N/2 + 1$ Spektrallinien zu berechnen, also die Frequenz von $0..N/2$ laufen zu lassen. Die Frequenz $f=0$ entspricht dabei dem Gleichanteil des Signals, die Frequenz $f=1$ stellt die *Grundschwingung* dar, die höheren Frequenzen bezeichnet man als *Harmonische*. Sie sind ganzzahlige Vielfache der Grundfrequenz.

Natürlich bietet es sich an, die gespiegelten Frequenzen gleich ganz wegzulassen, um den Berechnungsaufwand weiter zu verringern. Zumal hiermit auch kein Informationsverlust einhergeht. In den weiteren Beispielen werden daher nur noch $N/2 + 1$ Spektrallinien berechnet und dargestellt.

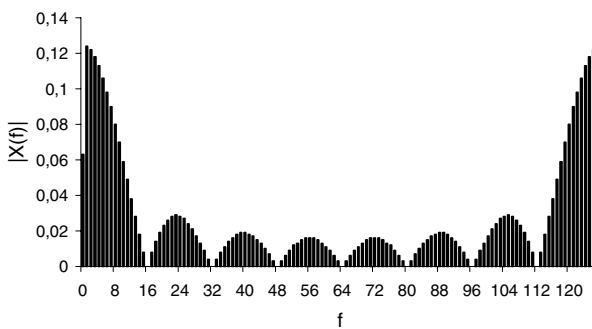
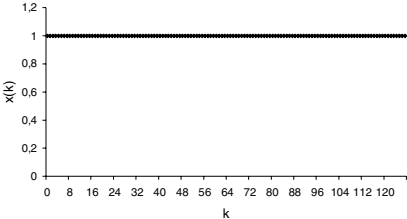
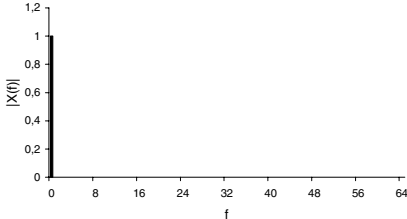
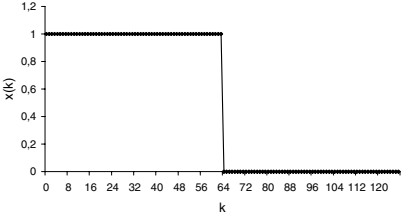
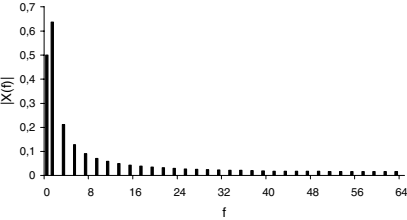
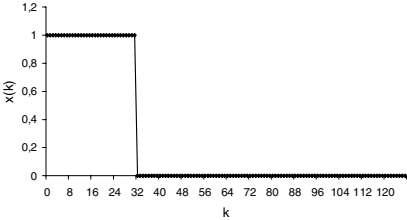
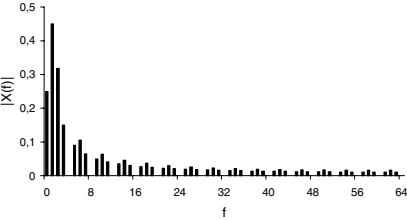
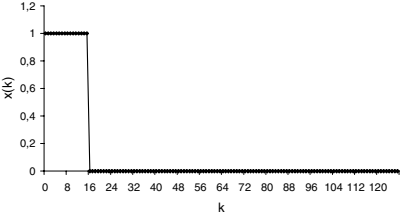
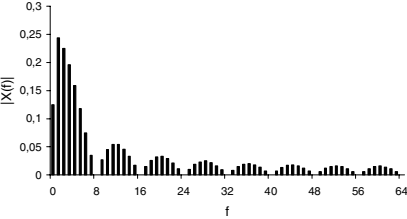
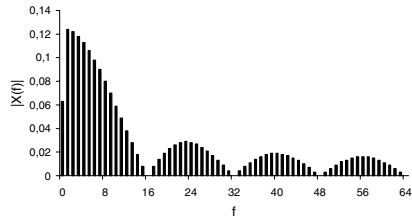
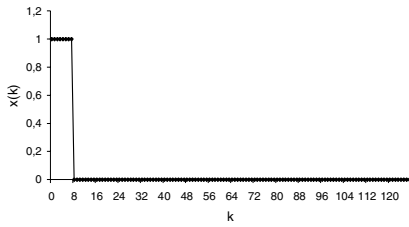


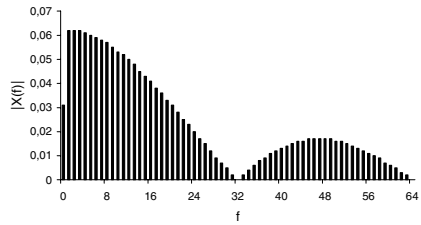
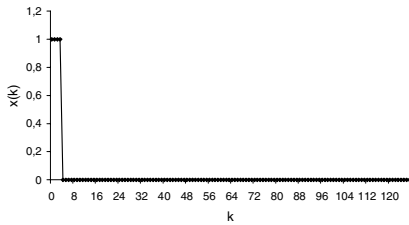
Abb. 1.8. Amplitudenspektrum des Rechteckimpulses

Tabelle 1.4. Amplitudenspektren verschiedener Pulsweiten

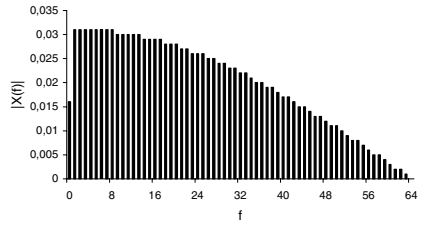
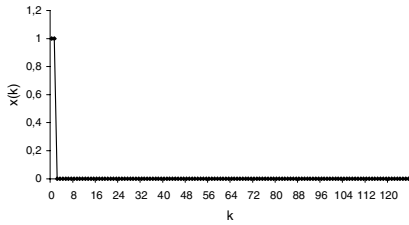
Zeitbereich $x(k)$ mit $k = 0..127$	Spektralbereich $X(f)$ mit $f = 0..64$
	
1/1 Puls	
	
1/2 Puls	
	
1/4 Puls	
	
1/8 Puls	



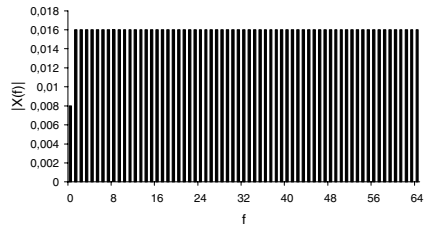
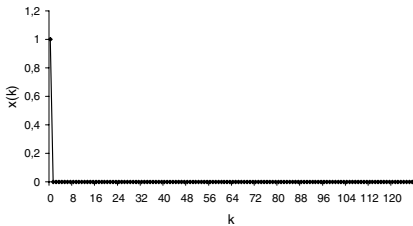
1/16 Puls



1/32 Puls



1/64 Puls



1/128 Puls

Eine wichtige Erkenntnis, die aus den Amplitudenspektren dieser Pulssignale ersichtlich wird, ist die zunehmende Ausdehnung der Hülle der Frequenzamplituden bei Abnahme der relativen Pulsweite. Gleichzeitig wird

das Spektrum deutlich flacher mit immer kleineren Amplituden für alle Harmonischen (beachte Skalierung), weil ein schmalerer Puls auch weniger Energie besitzt. Im Grenzfall einer gegen Null strebenden Pulsdauer besitzen alle Harmonischen die gleichen Amplituden, die aber sehr klein werden und damit ebenfalls gegen Null streben. Würde im Gegenzug die Größe der Pulse entsprechend der Verringerung der Pulsdauer erhöht, so daß das Produkt aus Pulsdauer und Pulshöhe konstant bleibt ($\Delta\tau \cdot h = \text{const}$), dann bleiben auch die Amplituden der Harmonischen konstant.

Die Aufweitung des Frequenzbereiches beschränkt sich nun keineswegs nur auf Pulse, sondern gilt allgemein für zeitlich stark lokalisierte Signale (z. B. steile Flanken) und läßt sich leicht interpretieren. Ursache hierfür sind die zeitlich unendlich ausgedehnten trigonometrischen Aufbaufunktionen. Sie ermöglichen die Signaldarstellung durch gegenseitige Verstärkung und Auslöschung, also durch konstruktive und destruktive Interferenz. Bei sehr scharfen zeitlichen Lokalisationen sind aber auch sehr hochfrequente Aufbaufunktionen notwendig, was schließlich zu der dargestellten spektralen Aufspreizung entlang der Frequenzachse führt. Wie am Beispiel eines immer schmaler werdenden Impulses gezeigt werden konnte, entartet das Signalspektrum auf diese Weise sogar zu einer Konstanten.

1.6 Der Abtastvorgang

Bei den zuvor eingeführten Impulsen führt eine Grenzwertbetrachtung der Pulsdauer mit $\Delta\tau \rightarrow 0$ (bei gleichzeitig konstanter Impulsfläche) zur δ -Funktion, die in der Literatur als *Delta-Impuls* oder *Dirac-Impuls* bekannt ist. Auf eine exakte mathematische Herleitung soll an dieser Stelle verzichtet werden. Der Delta-Impuls zeichnet sich durch die folgenden Eigenschaften aus:

$$\delta(t) = \begin{cases} 1 & (t = 0) \\ 0 & (t \neq 0) \end{cases} \quad (1.19)$$

$$\int_{-\infty}^{\infty} \delta(t) dt = 1 \quad (1.20)$$

$$\int_{-\infty}^{\infty} \delta(t) \cdot x(t) dt = x(0) \quad (1.21)$$

Eine besondere Bedeutung kommt vor allem der letzten Gleichung zu, welche die sogenannte *Ausblendeigenschaft* des Delta-Impulses beschreibt. Dank dieser Eigenschaft ist es leicht möglich, den Wert zeitkontinuierli-

cher Funktionen zu beliebigen Zeitpunkten zu erfassen. Macht man das in regelmäßigen Abständen, erhält man im Ergebnis eine Folge verschobener Impulse, deren jeweilige Höhe gleich dem entsprechenden Funktionswert von $x(t)$ ist (siehe auch Abb. 1.9).

$$x(k) = \sum_{k=0}^{\infty} \delta(t - k \cdot \Delta t) \cdot x(t) \quad (1.22)$$

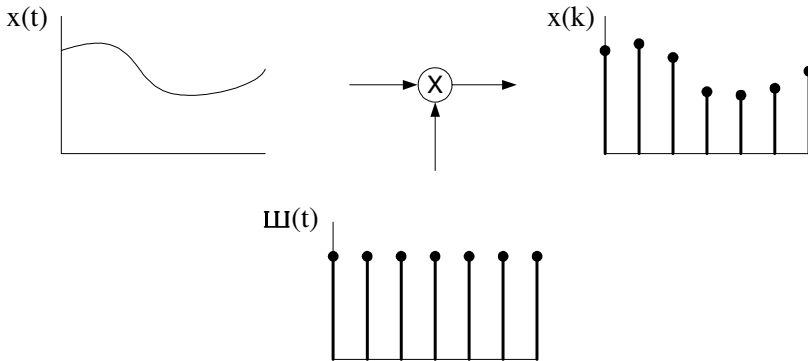


Abb. 1.9. Abtastung einer zeitkontinuierlichen Funktion

Die meßtechnische Erfassung einer Zeitfunktion wird *Abtastung* genannt. Hierbei werden dem zeitkontinuierlichen Funktionsverlauf in üblicherweise festen zeitlichen Abständen Werte entnommen, welche in ihrer Gesamtheit die zeitdiskrete Abtastfolge bilden. Die periodische Fortsetzung von $\delta(t)$ bezeichnet man als *Kammfunktion* $III(t)$ oder auch als *Delta-Kamm*. Aus der Funktion $x(t)$ wird somit durch Multiplikation mit einem solchen Delta-Kamm die abgetastete Funktion $x(k)$.

Streng genommen werden deren Werte bis zum nächsten Abtastzeitpunkt als konstant angenommen, weshalb man in diesem Fall auch von einem *Haltevorgang* nullter Ordnung spricht (bei einem Halteglied erster Ordnung sind die Abtastwerte durch Geraden verbunden, bei einem Halteglied zweiter Ordnung durch Parabeln, usw.). Am Ausgang eines Analog-Digital-Umsetzers (ADU), der die technische Realisierung eines solchen Abtast- und Haltegliedes darstellt, liegt tatsächlich also eine *Treppenfunktion* vor. Obwohl mathematisch sehr leicht möglich, tritt die Aufspaltung von Abtastvorgang und Haltevorgang in existierenden technischen Systemen meist jedoch nicht explizit in Erscheinung. Gleichwohl sollte man sich bewußt sein, daß beide Vorgänge auf das resultierende Spektrum wirken. Wir wollen solche Überlegungen aber vernachlässigen und uns im

weiteren Verlauf nur mit reinen Abtastfolgen (ohne Haltevorgang) beschäftigen. Das Spektrum einer Kammfunktion zeigt die nachfolgende Abbildung.

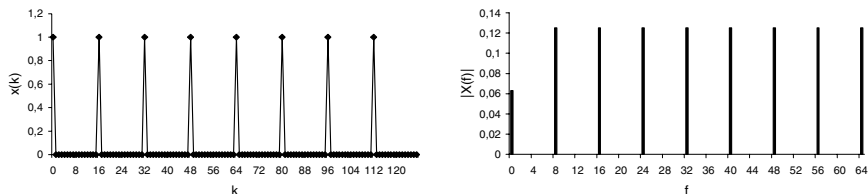


Abb. 1.10. Kammfunktion und ihr Spektrum

Interessanterweise ist das Spektrum einer Kammfunktion wiederum eine Kammfunktion, lediglich der Maßstab ändert sich. Je weiter die einzelnen Pulse im Zeitbereich auseinander liegen, desto näher rücken sie im Spektrum zusammen und umgekehrt. Dieser Umstand steht im Einklang mit den in Tabelle 1.4 dargestellten Impulsspektren, bei denen der erste Fall einem unendlich kleinen zeitlichen Abstand und der letzte Fall einem unendlich großen zeitlichen Abstand zwischen den einzelnen Pulsen entspricht. Wegen der speziellen Eigenschaften von $III(t)$ führt der Abtastvorgang eines Zeitsignals zu einer periodischen Fortsetzung seines Spektrums im Frequenzbereich. Da die Kammfunktion durch die Fouriertransformation wieder in sich selbst übergeht, bildet sie eine Eigenfunktion der Fouriertransformation. Eine Gemeinsamkeit, die sie mit der bekannten Gaußfunktion teilt.

Den Haltevorgang zwischen zwei Abtastungen kann man sich am besten als Impuls vorstellen, dessen Höhe dem ersten Abtastwert entspricht und dessen Weite mit dem Abtastintervall (= Kehrwert der Abtastfrequenz f_a) identisch ist. Folglich ist auch das resultierende Spektrum identisch mit den oben betrachteten Impulsspektren, abhängig von der Dauer des Haltevorgangs. Als Vorgriff auf spätere Abschnitte sei hier erwähnt, daß der Haltevorgang Tiefpaßeigenschaften besitzt. Sein Spektrum wird mathematisch durch eine Gleichung folgenden Typs beschrieben:

$$X(f) = \frac{\sin(\pi f / f_a)}{\pi f / f_a} \quad (1.23)$$

Die Funktion $\sin(x)/x$ ist von grundlegender Bedeutung in der digitalen Signalverarbeitung, und auch weit darüber hinaus bekannt. In der Optik beschreibt sie die Intensitätsverteilung des Lichtes nach dem Durchtritt durch einen schmalen Spalt. Ihrem physikalischen Ursprung entsprechend

wird sie daher *Spaltfunktion* genannt, manchmal auch sinc-Funktion oder einfach nur si-Funktion.

1.7 Das Abtasttheorem

Obwohl der Umstand, daß die reelle DFT aus den N Abtastwerten eines Signals genau $N/2 + 1$ Spektrallinien bestimmt, auf den ersten Blick beinahe nebensächlich erscheint, ist er für die digitale Signalverarbeitung von fundamentaler Bedeutung. Es leuchtet unmittelbar ein, je höher die Abtastrate (d. h. die Zahl der Abtastwerte je Periode) für eine Messung gewählt wird, desto mehr Details des Signals werden auch erfaßt. Dieser eigentlich doch selbstverständliche Zusammenhang kann nun aufgrund der oben gewonnenen Erkenntnisse genauer formuliert werden. Ein abgetastetes Signal enthält demnach nur Frequenzen bis zur halben Abtastfrequenz.

$$\begin{aligned} \text{höchste Signalfrequenz} &\leq \frac{1}{2} \cdot \text{Abtastfrequenz} \\ \text{Abtastfrequenz} &\geq 2 \cdot \text{höchste Signalfrequenz} \end{aligned} \quad (1.24)$$

Im Umkehrschluß ergibt sich damit, daß die Abtastfrequenz bei der Meßwerterfassung mindestens doppelt so hoch gewählt werden muß, wie die höchste Signalfrequenz. Diese Bedingung ist heute ganz allgemein als *Abtasttheorem* bekannt und wurde 1948 von Claude Elwood Shannon (1916–2001) in seiner Arbeit „A Mathematical Theorie of Communication“ formuliert. Die höchste in einem Signal enthaltene Frequenz wird auch *Nyquistfrequenz* genannt. Wie bereits erwähnt, führt die Abtastung eines Signals zu einer periodischen Fortsetzung des Signalspektrums im Frequenzbereich, wobei der Abstand der einzelnen Spektren gerade durch die Abtastfrequenz gegeben ist. Wird diese nun zu klein gewählt (*Unterabtastung*), kommt es zu einer unerwünschten Überlappung der periodischen Signalspektren, was schließlich zu einer Signalverfälschung im Zeitbereich führt. Solche durch den Abtastvorgang selbst erzeugten Fehlfrequenzen nennt man *Alias-Frequenzen*, den Vorgang entsprechend *Aliasing*. Abbildung 1.11 verdeutlicht den Effekt.

In solchen Fällen ist die ursprüngliche Information unwiederbringlich verloren und läßt sich auch nicht zurückgewinnen. Noch deutlicher wird das, wenn die Abtastfrequenz genau der Grundschiwingung des abgetasteten Signals entspricht. Die Abtastfolge entartet dann zu einem konstanten Wert, abhängig von der Phasenlage des Abtastzyklus. Praktisch ist aber auch die Mindestabtastrate gemäß dem Abtasttheorem von Shannon nicht ausreichend, was an den nichtidealen Eigenschaften der digitalen Filter

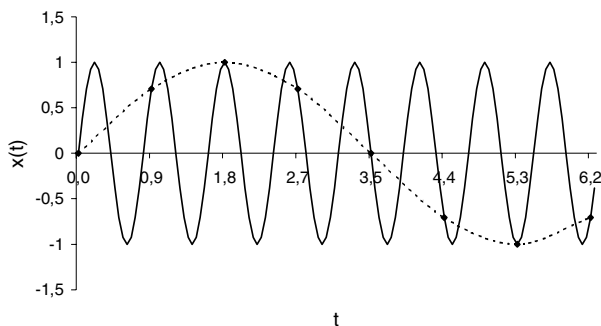


Abb. 1.11. Scheinbarer zeitlicher Signalverlauf bei Unterabtastung

innerhalb der Signalverarbeitungskette liegt. Oft wählt man daher die tatsächliche Abtastfrequenz deutlich höher. Durch eine solche *Überabtastung* (*oversampling*) steigen dann natürlich auch die Anforderungen an die verwendete Hardware.

In der Regel sind die in einem Signal enthaltenen Frequenzen vor der Messung nicht vollständig bekannt, was die Wahl einer geeigneten Abtastfrequenz natürlich erschwert. Wie soll man eine Frequenz messen, d. h. doppelt hoch abtasten, die man nicht einmal kennt? Dieses Dilemma läßt sich nur durch eine künstliche Begrenzung der Signalfrequenzen auf einen für die Messung interessanten Bereich lösen, mit Hilfe eines Anti-Aliasing-Filters. Dessen Frequenz bestimmt dann die gewählte Abtastrate.

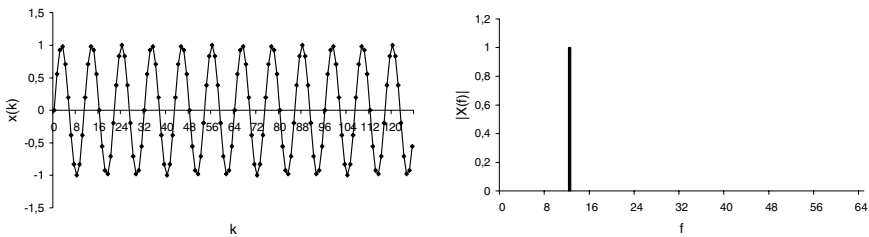
1.8 Leakage

Obwohl nicht ausdrücklich erwähnt, wurde in den bisherigen Beispielen stets davon ausgegangen, daß der Abtastvorgang exakt auf die Periodizität des zu untersuchenden Signals abgestimmt ist. Dabei kann es in der Praxis durchaus vorkommen, daß das abgetastete periodische Signal nicht genau aus einer ganzen Anzahl von Zyklen besteht. Dies ist unmittelbar einzusehen, da ja die Frequenzanalyse eines Signals erst genauere Kenntnisse über dessen Periodendauer bzw. dessen Grundfrequenz erbringen soll. Im Ergebnis macht sich dieser Umstand als eine deutliche Abweichung vom idealen Spektrum bemerkbar und es treten bei der DFT Frequenzkomponenten auf, die im Signal eigentlich gar nicht enthalten sind. Man bezeichnet diese spektrale Spreizung als *Leakage*. Ein passendes Testsignal kann einfach erzeugt und mit der entsprechenden Harmonischen verglichen werden.

```

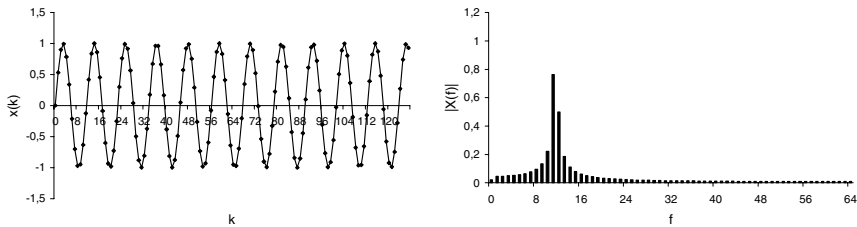
class Spektrum
{
    ...
    public void ErzeugeTestsignal_2()
    {
        for (int k=0; k<Abtastwerte; k++)
        {
            x[k] = Math.Sin(2*PI*12*0.95*k/Abtastwerte);
        }
    }
    ...
}

```



$$T_{\text{Abtast}}/T_{\text{Signal}} = 12$$

Abb. 1.12. Amplitudenspektrum bei ganzzahliger Zyklenzahl



$$T_{\text{Abtast}}/T_{\text{Signal}} = 11,4$$

Abb. 1.13. Leakage bei nicht ganzzahliger Zyklenzahl

Worin kann nun die Ursache für diesen Effekt gesehen werden? Sowohl die DFT als auch die FFT verarbeiten einen zeitlich endlichen Signalausschnitt für die Frequenzanalyse. Dessen Länge ergibt sich dabei durch das verwendete Zeitfenster von N Abtastwerten. Nach der Theorie sollte sich das unendlich fortgesetzte Zeitsignal genau dann ergeben, wenn man das betrachtete Zeitfenster mehrfach hintereinander reiht. Fällt eine ganze Zahl von Signalzyklen in das Zeitfenster, funktioniert das auch tatsächlich. Ist das jedoch nicht der Fall, entstehen an den Intervallgrenzen ungewollt

Signalsprünge, die es ursprünglich nicht gab. Sie sind verantwortlich für die entstehende spektrale Aufspreizung.

Grundsätzlich ist bei der DFT die Zahl der Abtastwerte frei wählbar, wodurch sich Diskontinuitäten im Signalverlauf aufgrund der Fensterlänge vermeiden lassen. Vorausgesetzt man kennt die exakte Periodendauer, woher auch immer. Etwas anders sieht es bei der FFT aus, wo die Fensterlänge in der Regel eine Zweierpotenz ist. Hier müssen andere Methoden greifen, um diesen Effekt zu beseitigen oder wenigstens spürbar zu dämpfen. Da die Signalsprünge erst an den Rändern des Zeitfensters auftreten, liegt der Versuch nahe, an diesen Stellen durch geeignete Maßnahmen einen kontinuierlichen Signalverlauf zu erzwingen. Angenommen man drückt das Signal an beiden Enden langsam auf Null, unabhängig von seinem tatsächlichen Verlauf, so wären die Übergänge angepaßt und die Diskontinuität beseitigt. Am einfachsten läßt sich das erreichen, indem man das Zeitsignal zusätzlich mit einem entsprechend geformten Fenster multipliziert. Dieser Vorgang wird *Fensterung* genannt und verändert natürlich auch das Signalspektrum. Nicht vergessen darf man aber hierbei, daß dies selbstverständlich auch für das Rechteckfenster gilt, mit dem wir bisher immer eine Anzahl von Abtastwerten aus dem unendlich angenommenen Signalverlauf betrachtet haben. Doch hierzu später mehr. In der digitalen Signalverarbeitung sind im Laufe der Zeit eine ganze Reihe von Fensterfunktionen entwickelt worden, die auf unterschiedliche Randbedingungen hin optimiert sind. Ein typischer Vertreter ist das *Hanning-Fenster* (manchmal auch von-Hann-Fenster). Wir wollen mit M die gewählte Anzahl von Abtastwerten pro Signalperiode bezeichnen.

$$w(k) = 0,5 - 0,5 \cdot \cos\left(\frac{2\pi k}{M}\right) \quad (1.25)$$

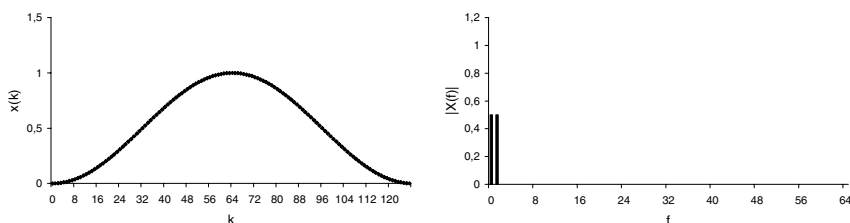


Abb. 1.14. Das Hanning-Fenster und sein Linienspektrum

Die Multiplikation des Signals mit der neuen Fensterfunktion im Zeitbereich führt zu einer *Faltung* des Signalspektrums mit dem Spektrum des jeweils verwendeten Fensters im Frequenzbereich. Es ist üblich für die

Faltung, die im Abschnitt über digitale Filter noch näher erläutert wird, den symbolischen Rechenoperator $*$ zu verwenden.

$$x(k) \cdot w(k) \longleftrightarrow X(f) * W(f) \quad (1.26)$$

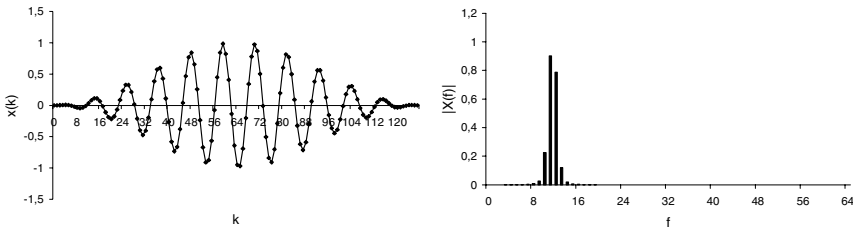


Abb. 1.15. Signalverlauf und Linienspektrum nach zusätzlicher Fensterung

Tatsächlich führt die Wahl einer geeigneten Fensterfunktion zu einer deutlichen Reduzierung der spektralen Aufspaltung, wie am resultierenden Signalspektrum zu erkennen ist. Bedingt durch die Fensterform und die damit zusammenhängende Dämpfung der Signalamplitude zu den Rändern hin, kommt es auch zu einer Dämpfung der Spektrallinien, die nach der Fensterung wieder ausgeglichen werden muß. Allgemein ist die Dämpfung umgekehrt proportional zur Fläche unterhalb der Fensterfunktion. Im vorliegenden Fall (Dämpfung = 2) müssen wir die Höhen der Spektrallinien mit Zwei multiplizieren, um die ursprünglichen Verhältnisse wiederzugeben. Andere Fensterfunktionen erfordern, entsprechend ihrem Gleichanteil, auch andere Korrekturwerte.

Die offensichtliche Verbesserung im Ergebnis könnte nun zu der Schlußfolgerung verleiten, es sei sinnvoll immer andere Fensterfunktionen als das gewöhnliche Rechteckfenster zu benutzen. Dies kann jedoch nicht grundsätzlich gelten. Wie der nächste Abschnitt zeigt, ist die Wahl einer optimalen Fensterfunktion nur unter Berücksichtigung der Beschaffenheit des zu untersuchenden Signals möglich.

1.9 Nichtstationäre Signale – Die Grenzen der DFT

Die in den vorangegangenen Abschnitten vorgestellte DFT eignet sich zur Frequenzanalyse von zeitdiskreten Abtastsignalen. In der Praxis kann es dabei durchaus vorkommen, daß sich der Frequenzbereich eines Signals innerhalb kurzer Zeit ändert. In elektrischen Netzwerken kommt es bei Schaltvorgängen aufgrund der vorhandenen Elemente (Induktivitäten, Kapazitäten) zu Ausgleichsvorgängen. Die dabei auftretenden *Transienten*

stellen Frequenzanteile dar, die sich deutlich vom gewöhnlichen Signalverlauf abgrenzen. Sie sind mit einer normalen DFT kaum zu erfassen. Das folgende Beispiel soll die Grenzen der bisherigen Frequenzanalyse deutlich machen. Hierzu wird ein geeignetes Testsignal erzeugt.

```
class Spektrum
{
    ...
    public void ErzeugeTestsignal_3()
    {
        for (int k=0; k<Abtastwerte; k++)
        {
            if (k < Abtastwerte/2)
            {
                x[k] = Math.Sin(2*PI*52*k/Abtastwerte);
            }

            if (k >= 3*Abtastwerte/4 &&
                k < (3*Abtastwerte/4 + Abtastwerte/16))
            {
                x[k] = Math.Sin(2*PI*23*k/Abtastwerte);
            }
        }
    }
    ...
}
```

Zunächst besteht unser erzeugtes Testsignal aus einer hochfrequenten Sinusschwingung (52. Harmonische) über die erste halbe Abtastperiode. In der Mitte der zweiten Hälfte kommt noch eine kurze (1/16 Periode) Schwingung mit der 23-fachen Grundfrequenz hinzu.

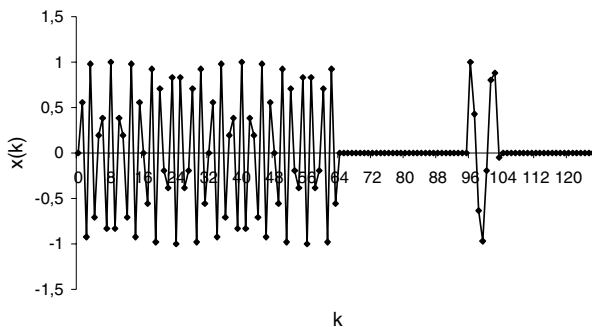


Abb. 1.16. Testsignal mit Frequenzanteilen unterschiedlicher Dauer

Reale Transienten treten meist in Form von exponentiell abklingenden, hochfrequenten Schwingungen auf. Unser Signal weicht hiervon deutlich ab, was jedoch nicht weiter stört. Als Ergebnis der DFT ergibt sich dann das folgende Spektrum.

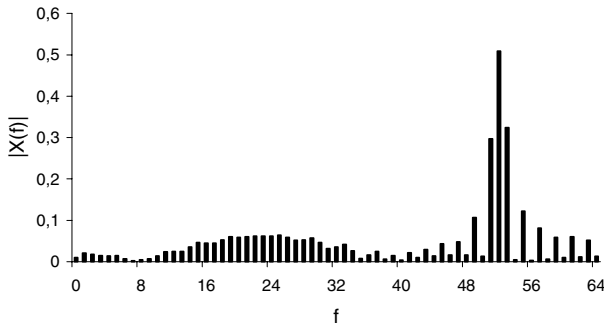


Abb. 1.17. Amplitudenspektrum des Testsignals

Deutlich zu erkennen sind die Anteile der 52. Harmonischen. Dagegen geht die kurzlebige Schwingung der 23. Harmonischen im Grundrauschen des Spektrums beinahe völlig unter, obwohl dessen Amplitude genauso groß ist, wie die der ersten Frequenz. Jedoch ist das Intervall in dem sie existiert nur sehr kurz, wodurch ihr Frequenzanteil herausgemittelt wird. Zusätzlich macht sich der Leakage-Effekt bemerkbar. Obwohl doch zeitweise vorhanden, ist im Amplitudenspektrum überhaupt nicht erkennbar, wann die jeweiligen Frequenzen auftreten oder wie lange sie wirksam sind. Gerade diese fehlende Information kann aber in bestimmten Anwendungen von erheblichem Nutzen sein.

1.10 Die Zeit-Frequenz-Analyse

Wie wir bisher gesehen haben, ist die Berechnung der spektralen Zusammensetzung eines Signals ein geeignetes Verfahren zu dessen Charakterisierung. Als besonders anschaulich erweist sich hierbei die grafische Darstellung des Amplitudenspektrums. Leider geht diese Anschaulichkeit verloren, wenn sich die spektrale Zusammensetzung des Signals innerhalb des zu untersuchenden Zeitraumes ändert. In der Realität kommt das sehr häufig vor, so zum Beispiel bei Sprachsignalen. Zwar kann man auch in solchen Fällen ein Gesamtspektrum mit den bisher vorgestellten Methoden berechnen, eine gegebenenfalls erforderliche zeitliche Zuordnung einzelner

Frequenzanteile ist aber nicht möglich und kurzlebige Komponenten verschwinden unter Umständen ganz.

Das Problem entsteht dadurch, daß die DFT und die FFT nur ein einziges Zeitfenster verwenden, welches sich immer über das gesamte Abtastintervall erstreckt. Für die Erfassung stark lokalisierter und damit hochfrequenter Signalkomponenten werden entsprechend hochfrequente Cosinus- und Sinusanteile herangezogen. Deren Periodendauer $T = 1/f$ ist aber deutlich kleiner als das Zeitfenster der DFT, wodurch es unmöglich ist, den Zeitpunkt ihres Auftretens eindeutig festzulegen (genau genommen ist die zeitliche Zuordnung auf sehr unanschauliche Weise im Phasenspektrum verschlüsselt). Es liegt daher nahe, das Problem durch eine verbesserte Lokalisierung der trigonometrischen Aufbaufunktionen zu entschärfen. In der praktischen Umsetzung sieht das folgendermaßen aus. Anstatt nur ein einziges großes Zeitfenster zu verwenden, wird das Zeitsignal in mehrere kurze Abschnitte aufgeteilt, innerhalb derer es als quasistationär angesehen werden kann. Zu jedem dieser Abschnitte wird anschließend ein separates Spektrum berechnet. Als Ergebnis erhält man auf diese Weise eine Folge von Einzelspektren, die in ihrer Gesamtheit die Dynamik des Signals widerspiegeln. Man bezeichnet dieses Vorgehen als *Kurzzeit-Spektralanalyse* oder kurz als *STFT* (Short-Time-Fourier-Transform). Eine einfache Implementierung könnte so aussehen:

```
class Spektrum
{
    ...
    const int MaxIntervalle = 32;
    ...
    public double[,] KS = new double[MaxIntervalle,
                                     Abtastwerte];
    ...
    public void ZeitFrequenzAnalyse(int Intervalle)
    {
        // über alle Teilintervalle
        for (int dt=0; dt<Intervalle; dt++)
        {
            // über alle Frequenzen
            for (int f=0; f<Abtastwerte; f++)
            {
                Re[f] = 0;
                Im[f] = 0;

                // über alle Abtastwerte des Teilintervalls
                for (int k=(Abtastwerte/Intervalle)*dt;
                     k<(Abtastwerte/Intervalle)*(1+dt); k++)
```

```

    {
        Re[f] += x[k]*Math.Cos(2*PI*f*k/Abtastwerte);
        Im[f] += x[k]*Math.Sin(2*PI*f*k/Abtastwerte);
    }
    Re[f] *= 2.0/Abtastwerte;
    Im[f] *= 2.0/Abtastwerte;

    KS[dt, f] = Math.Sqrt(Re[f]*Re[f] + Im[f]*Im[f]);
}
Re[0] /= 2.0; // Korrektur Gleichanteil
KS[dt, 0] = Re[0];
}
...
}

```

Grundsätzlich dürfen sich die aufeinanderfolgenden Signalabschnitte natürlich auch überlappen, was mit der Vorstellung korrespondiert, daß unser Zeitfenster kontinuierlich über das Signal hinweggleitet. Für unser Testsignal erhält man bei einer zeitlichen Auflösung von 16 Teilintervallen (ohne Überlappung) die nachfolgende Zeit-Frequenz-Darstellung, die auch *Spektrogramm* genannt wird. Um den Charakter eines Spektrums zu erhalten, wurde die Frequenz f als Abszisse und die Zeit k als Ordinate gewählt. Grundsätzlich ist aber natürlich auch die umgekehrte Darstellung möglich.

Man erkennt nicht nur die im Signal enthaltenen Frequenzanteile, sondern kann diese jetzt auch eindeutig zeitlich zuordnen. Schaut man sich auch die dreidimensionale Darstellung des Spektrogramms an, erkennt man zudem sofort, daß kurzlebige Signalanteile nicht einfach rausgemittelt werden, sondern aufgrund der bei der STFT verwendeten kurzen Teilintervalle, ihrer Amplitude entsprechend, gleichfalls Berücksichtigung finden.

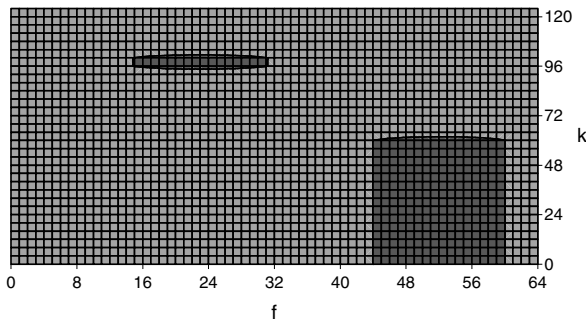


Abb. 1.18. Spektrogramm des Testsignals mit 16 Teilintervallen

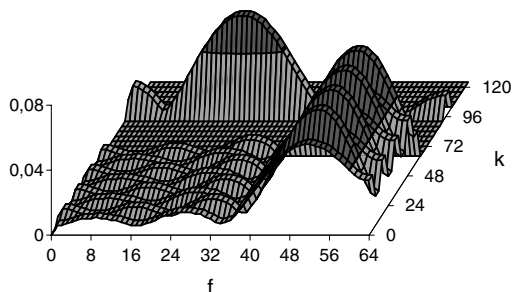


Abb. 1.19. Spektrogramm in 3D-Ansicht

Wie schon im Zusammenhang mit dem Leakage-Effekt gezeigt, führt das bloße Ausschneiden eines Signalabschnittes, das einer Multiplikation des Signals mit einem Rechteckfenster entspricht, unter Umständen zu einem Verwischen des Frequenzspektrums, welches durch die Wahl einer anderen Fensterfunktion gemildert werden kann. Es kann sich daher auch beim Kurzzeit-Spektrum als zweckmäßig erweisen auf andere Fensterfunktionen zurückzugreifen. Hat man es mit Transienten oder anderen kurzlebigen Anteilen im Signalverlauf zu tun, ist aber Vorsicht geboten. Solche Erscheinungen können natürlich auch am Rand des betrachteten Intervalls liegen und werden bei Verwendung einer ungeeigneten Fensterfunktion unter Umständen ausgeblendet, wodurch wichtige Frequenzinformationen verloren gehen. Dies gilt übrigens auch für die normale DFT oder FFT. Verwendet man ein gleitendes Fenster, beispielsweise mit einer 50% Überlappung, läßt sich die unerwünschte Ausblendung am Rand leicht vermeiden. Signalanteile, die in einem Intervall am rechten Rand auftreten, liegen dann im nächsten Intervall genau in der Mitte und werden somit voll berücksichtigt.

Es leuchtet unmittelbar ein, daß die gewählte zeitliche Rasterung bei der STFT erheblichen Einfluß auf die zeitliche Auflösung hat. Wählt man die Teilintervalle sehr klein, kann man zwar stark lokalisierte (und damit hochfrequente) Ausprägungen zeitlich gut auflösen, niedrige Frequenzanteile mit entsprechend großer Periodendauer werden dann aber nur noch ungenügend oder gar nicht mehr erfaßt. Dies läßt sich zwar durch die Verwendung größerer Teilintervalle vermeiden, worunter dann aber wieder die Auflösung zeitlich lokaler Ereignisse leidet. Ähnlich wie bei Heisenbergs Unschärfetheorie in der Physik, hat man es hier unvermeidbar mit einer *Zeit-Frequenz-Unschärfe* zu tun. Möchte man mehr zeitliche Details, muß man auf Frequenzinformationen verzichten und umgekehrt. Damit ist klar, daß die Wahl einer bestimmten Fensterfunktion und einer geeigneten Zeitfensterlänge unbedingt problemspezifisch erfolgen muß und immer nur auf den jeweiligen Anwendungsfall optimal abgestimmt werden kann.

Als zweites Beispiel für ein Signal mit zeitlich veränderlicher Frequenz betrachten wir den *Gleitsinus*, der auch als *Chirp* („Zwitschern“) bekannt ist. Wegen seiner linear ansteigenden Momentanfrequenz findet er unter anderem in der Meßtechnik bei der Systemanalyse, oder in der Radartechnik Anwendung. Abbildung 1.20 zeigt die besondere Charakteristik dieses Signals. Als Ton hörbar gemacht ähnelt es dem Zwitschern eines Vogels, woraus sich sein Name ableitet.

```
class Spektrum
{
    ...
    public void ErzeugeTestsignal_4()
    {
        double f = 0;

        for (int k=0; k<Abtastwerte; k++)
        {
            x[k] = Math.Sin(2*PI*f*k/Abtastwerte);
            f += 0.25;
        }
    }
}
```

Die scheinbare Amplitudenmodulation in Abb. 1.20 resultiert aus ungünstig verteilten Abtastpunkten. Wie nicht anders zu erwarten war, zeigt sich die stetig wachsende Signalfrequenz als steigende Gerade im Spektrogramm. Wählt man jedoch die Anzahl der Teilintervalle ungünstig, wie in Abb. 1.22, dann bleiben unter Umständen wesentliche Signaleigenschaften verborgen. Ein Rückschluß auf die tatsächliche Zeit-Frequenz-Charakteristik ist dann oft nur noch sehr schwer (oder überhaupt nicht mehr) möglich.

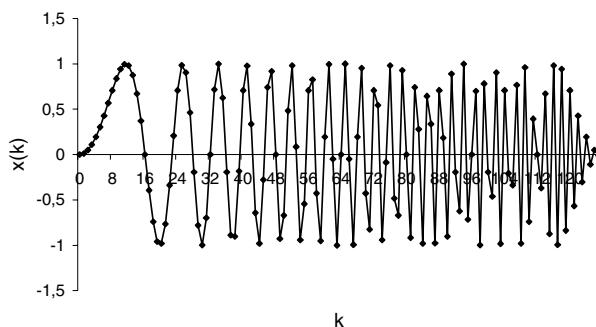


Abb. 1.20. Zeitverlauf des Chirp mit linear ansteigender Frequenz

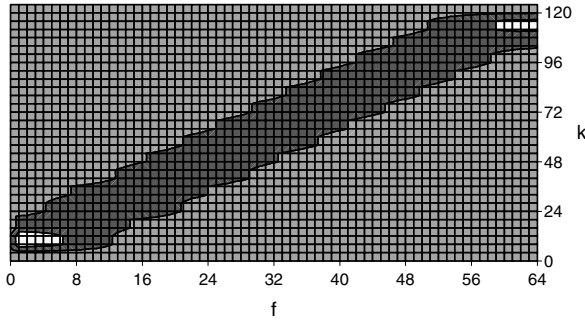


Abb. 1.21. Spektrogramm des Chirp mit 16 Teilintervallen

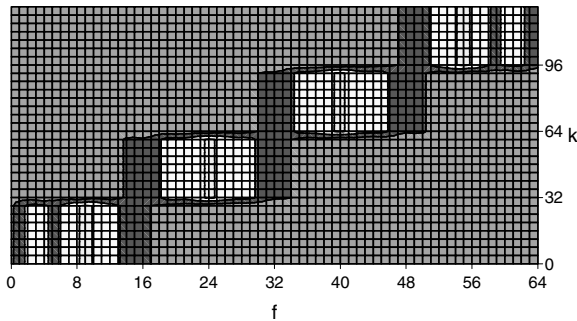


Abb. 1.22. Spektrogramm des Chirp mit 4 Teilintervallen

1.11 Digitale Filter

In der Signalverarbeitung setzt man *digitale Filter* hauptsächlich für die Rekonstruktion verrauschter Eingangssignale oder für die Trennung verschiedener Frequenzanteile ein. Grundsätzlich sind für beide Anwendungen auch analoge Filter in Form von elektronischen Schaltungen geeignet. Allerdings streuen die elektrischen Kenngrößen analoger Bauelemente sehr stark. Zusätzlich unterliegen sie einem Alterungsprozeß, der die Charakteristik analoger Filter im Laufe mehrere Betriebsjahre merklich verändert. Dagegen werden digitale Filter in der Regel als Softwareimplementierung eines entsprechenden Algorithmus realisiert, wodurch die Stabilität ihrer Eigenschaften immer gewährleistet ist, auch über viele Jahre hinweg. Um den Umfang dieses Kapitels nicht zu sprengen, werden ausschließlich lineare Filter betrachtet.

1.11.1 Frequenzselektive Eigenschaften

Digitale Filter haben die Aufgabe bestimmte Signalfrequenzen entweder zu verstärken oder zu dämpfen. Mit ihrer Eigenschaft auf verschiedene Frequenzen in unterschiedlicher Weise einzuwirken, sind sie leicht zu beschreiben. Es ist daher gängige Praxis, frequenzselektive Filter über ihr Frequenzverhalten zu klassifizieren. Der *Durchlaßbereich* ist derjenige Bereich, der von Frequenzen passiert werden kann, wogegen der *Sperrbereich* die blockierten Frequenzanteile kennzeichnet. Beide Bereiche sind durch einen sogenannten *Übergangsbereich* voneinander getrennt. Für eine saubere Trennung der Frequenzen, sollte dieser so schmal wie möglich ausfallen, seine Breite also gegen Null gehen. In der Realität ist dieses angestrebte Ideal ist aber nicht zu erreichen, weshalb man sich beim Filterentwurf gezwungenermaßen mit einem mehr oder weniger guten Kompromiß begnügt. Weitere Kenngrößen sind die *Bandbreite* eines Filters, sowie seine *Grenzfrequenz*. Beide Größen sind eng miteinander verknüpft und beschreiben den Frequenzbereich, der weitestgehend unverändert das Filter durchlaufen kann. Tatsächlich wird auch im Durchlaßbereich eine gewisse Dämpfung der Signalamplitude akzeptiert, die bei der Grenzfrequenz auf $1/\sqrt{2}$ (dies entspricht 70,7%) ihres ursprünglichen Wertes abgesunken ist. Die Amplitude des Frequenzganges wird auch *Verstärkung* genannt und stellt ebenfalls eine wichtige Kenngröße digitaler Filter dar. Sie ergibt sich aus dem Verhältnis der Ausgangsamplitude zur Eingangsamplitude des Filtersignals. Ist die Verstärkung größer als Eins, dann ist die Ausgangsamplitude größer als die Eingangsamplitude. Im umgekehrten Fall ist sie kleiner als Eins. In den meisten Anwendungen ist die Verstärkung im Sperrbereich erheblich kleiner als im Durchlaßbereich, typischerweise um mehrere Zehnerpotenzen. Um dennoch beide Bereiche auf einer Skala grafisch vernünftig darstellen zu können, wird oft die zunächst lineare Verstärkung in Dezibel (dB) umgerechnet.

$$Verstärkung_{dB} = 20 \cdot \log(Verstärkung) \quad (1.27)$$

Für den täglichen Gebrauch ist es zweckmäßig einige Umrechnungen zwischen dem linearen Amplitudenverhältnis und den entsprechenden Werten in Dezibel im Kopf zu haben. Eine Zusammenstellung der wichtigsten Wertepaare ist aus diesem Grund hier aufgelistet.

Tabelle 1.5. Logarithmisches (dB) und lineares Amplitudenverhältnis

-60 dB	0,001
-40 dB	0,01
-20 dB	0,1
-6 dB	0,5
-3 dB	0,707
0 dB	1
3 dB	1,414
6 dB	2
20 dB	10
40 dB	100
60 dB	1000

Die erwähnte Unterteilung frequenzselektiver Filter in funktionelle Gruppen orientiert sich am jeweils wirksamen Frequenzbereich, woraus sich schließlich auch die Benennung der verschiedenen Filtertypen ableitet. Es könne vier Grundtypen unterschieden werden, von denen jedoch zum Teil noch Varianten existieren.

Tiefpaß-Filter werden eingesetzt, um die tiefen Frequenzanteile eines Signals zu übertragen, wogegen die hohen Frequenzen oberhalb der Grenzfrequenz f_g gedämpft werden. Ihr Durchlaßbereich erstreckt sich von $f=0\text{Hz}$ bis $f=f_g$. Mit ihnen gelingt es hochfrequentes Rauschen vom Nutzsignal zu trennen. Die sehr häufig angewendete Mittelwertbildung von Meßwerten stellt eine einfache Tiefpaß-Filterung dar.

Hochpaß-Filter sind notwendig, um die hohen Frequenzanteile eines Signals zu übertragen. Tiefe Frequenzen unterhalb der Grenzfrequenz und auch der Gleichanteil werden dagegen gedämpft. Ihr Durchlaßbereich erstreckt sich von $f=f_g$ bis $f=\infty$. Die oft angewendete Differenzbildung von Meßwerten, zum Beispiel zur Erkennung von Änderungen im Signalverlauf, stellt eine einfache Hochpaß-Filterung dar.

Bandpaß-Filter sind in der Lage einen mittleren Frequenzbereich zu übertragen. Tiefe Frequenzen unterhalb der unteren Grenzfrequenz und hohe Frequenzen oberhalb der oberen Grenzfrequenz werden dagegen blockiert. Hierdurch bedingt, besitzen sie auch zwei Sperrbereiche. Verengt sich der Durchlaßbereich auf eine Frequenz, spricht man auch von einem *Resonanz-Filter*.

Bandsperrn bzw. *Sperr-Filter* dämpfen einen mittleren Frequenzbereich, wogegen tiefere und höhere Frequenzen übertragen werden. Sie besitzen also zwei Durchlaßbereiche. Verengt sich der Sperrbereich auf eine Frequenz, spricht man auch von einem *Notch-Filter*, manchmal auch Loch- oder Kerb-Filter.

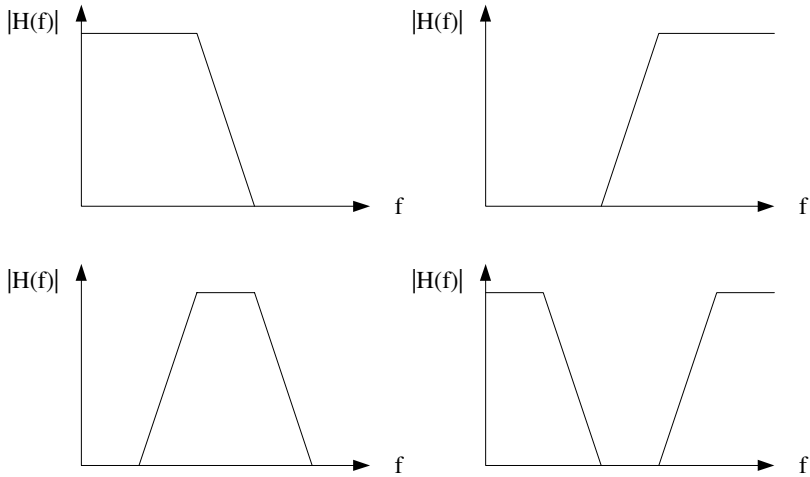


Abb. 1.23. Tiefpaß, Hochpaß, Bandpaß und Bandsperre (unten rechts)

1.11.2 Die z -Transformation

Lineare Filter lassen sich wahlweise durch ihr Verhalten im Zeitbereich oder im Frequenzbereich beschreiben. Beide Bereiche sind gleichermaßen geeignet, ein lineares Filter vollständig zu charakterisieren. Es sind nur jeweils verschiedene Betrachtungsweisen des selben Sachverhalts. Sind erst einmal durch die innerhalb einer Anwendung herrschenden Randbedingungen die Eigenschaften im Zeitbereich vorgegeben, ist damit auch das Verhalten im Frequenzbereich eindeutig festgelegt und kann berechnet werden (gilt auch umgekehrt).

Um diese Zusammenhänge besser beschreiben und auch verstehen zu können, kommt man spätestens an dieser Stelle leider nicht umhin, sich zumindest ein wenig mit dem mathematischen Fundament der digitalen Signalverarbeitung zu beschäftigen. Wie wir gesehen haben, führt die alleinige Betrachtung von Signalen im Zeitbereich zu einer unvollständigen Sicht der Dinge. Die Tatsache, daß so manche Signaleigenschaft erst im Frequenzbereich sichtbar wird, rechtfertigt den Einsatz von Frequenztransformationen. Als bekanntes und weit verbreitetes Werkzeug für zeitkontinuierliche Signale hat sich die Fourier-Transformation bewährt, die ein Sonderfall der *Laplace-Transformation* (abgekürzt L) ist. Für abgetastete Signale wurde die DFT vorgestellt. Sie kann ohne viel Aufwand als Algorithmus auf einem Computer implementiert werden, stellt aber, entsprechend ihrem kontinuierlichen Gegenstück, nur einen Sonderfall der allgemeineren z -Transformation (Z) dar. Somit bildet die z -Transformation das zeitdiskrete Pendant zur zeitkontinuierlichen Laplace-Transformation.

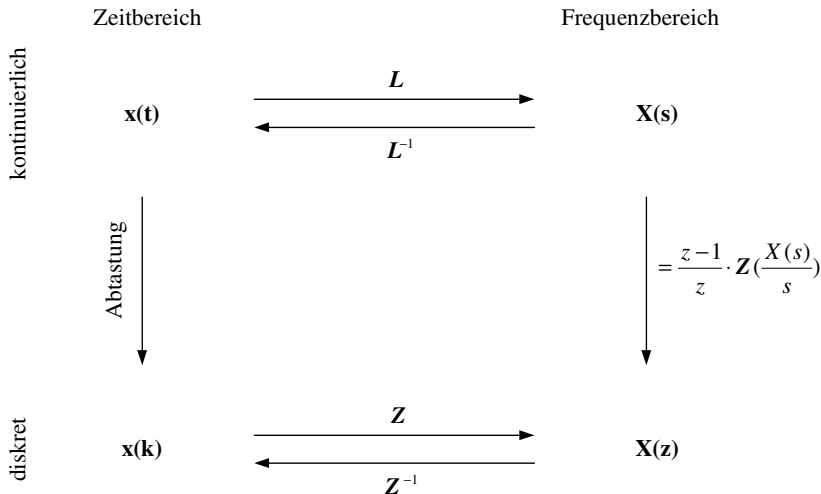


Abb. 1.24. Transformation zwischen Zeitbereich und Frequenzbereich

Die vorgestellten Verfahren wurden bisher ausschließlich auf Signale angewendet. Es ist nun an der Zeit, diese Selbstbeschränkung zugunsten des Begriffs *System* endlich aufzugeben. Als System kann eine beliebige Anzahl von Elementen gesehen werden, die miteinander in Wechselwirkung stehen. Üblicherweise lassen sich diese Beziehungen mit Hilfe von Signalen mathematisch beschreiben. Ein Digitalfilter stellt ein einfaches Signalverarbeitungssystem dar, welches in der Regel Teil eines größeren Gesamtsystems ist.

Das Bild verdeutlicht die Zusammenhänge zwischen Signalen im Zeitbereich und im Frequenzbereich bzw. den Übergang von den zeitkontinuierlichen Signalen hin zu zeitdiskreten Signalen. Es mag ein wenig überraschen, daß keine Umkehrung der Abtastung existiert. Dies steht scheinbar im Widerspruch zur allgemeinen Vorstellung, wonach Abtastfolgen natürlich auch wieder in zeitkontinuierliche Signale umgewandelt werden können. Schließlich gibt es Digital-Analog-Umsetzer (DAU), kleine elektronische Bausteine, die genau hierzu in der Lage sind. Allerdings handelt es sich bei einer solchen Rekonstruktion immer nur um eine mehr oder weniger gelungene Interpolation. Wie man sich leicht vorstellen kann, ist es immer möglich, durch die Stützstellen einer Abtastfolge unendlich viele kontinuierliche Signalverläufe zu legen. Ausgangspunkt für die weiteren Betrachtungen ist aber zunächst die Laplace-Transformation.

$$X(s) = \int_{t=-\infty}^{+\infty} x(t) \cdot e^{-st} dt \quad (1.28)$$

Mit Hilfe der Substitution $z = e^{sT}$ wird daraus schließlich die zweiseitige z -Transformation.

$$X(z) = \sum_{k=-\infty}^{+\infty} x(k) \cdot z^{-k} \quad (1.29)$$

$X(z)$ ist somit eine Funktion der komplexen Variablen z und kann in der komplexen Ebene dargestellt werden. Weiterhin gilt $s = \sigma + j\omega$.

$$\begin{aligned} z &= e^{sT} \\ z &= e^{\sigma} \cdot e^{j\omega T} \\ z &= r \cdot e^{j\omega T} \end{aligned} \quad (1.30)$$

Die hier aufgeführten Gleichungen ermöglichen die Transformation von Signalen aus der komplexen s -Ebene in die komplexe z -Ebene. Berücksichtigt man die verschiedenen möglichen Werte von $\sigma = \text{Re}(s)$, ergeben sich grundsätzlich drei unterschiedliche Bereiche.

- $\sigma = 0$ entspricht in der s -Ebene der Imaginärachse. Als Radius erhält man $r = e^0 = 1$. Die Imaginärachse der s -Ebene wird auf den Einheitskreis der z -Ebene abgebildet. Dies entspricht gerade der schon mehrfach verwendeten DFT.
- $\sigma > 0$ beschreibt die rechte offene s -Halbebene, was im z -Bereich einem Radius $r > 1$ entspricht. Damit wird die rechte s -Ebene also auf den außerhalb des Einheitskreises liegenden Bereich transformiert.
- $\sigma < 0$ ergibt einen Radius $r < 1$, was gleichbedeutend ist mit einer Abbildung der linken s -Halbebene in das innere des Einheitskreises der z -Ebene.

Eine ganz besondere Bedeutung kommt der s -Ebene in der Regelungstechnik bei Untersuchungen zur *Stabilität* zeitkontinuierlicher Systeme zu. Sie lassen sich im Zeitbereich mathematisch durch Differentialgleichungen beschreiben. Als Lösung dieser Differentialgleichungen ergeben sich für gewöhnlich Exponentialfunktionen. Es leuchtet unmittelbar ein, daß diese Exponentialfunktionen, und damit natürlich auch das System selbst, nur für negative Exponenten stabil sind. Somit gewährleisten die negativen Exponenten in den Lösungen der systembeschreibenden Differentialgleichung die Stabilität eines zeitkontinuierlichen Systems. In die s -Ebene transformiert, bilden sie die dann ebenfalls negativen Polstellen (Pole = Nullstellen des Nennerpolynoms) der gebrochen rationalen *Übertragungsfunktion* des Systems. Für zeitkontinuierliche Systeme kann daher festgehalten werden, daß diese genau dann stabil sind, wenn ihre Übertragungsfunktion

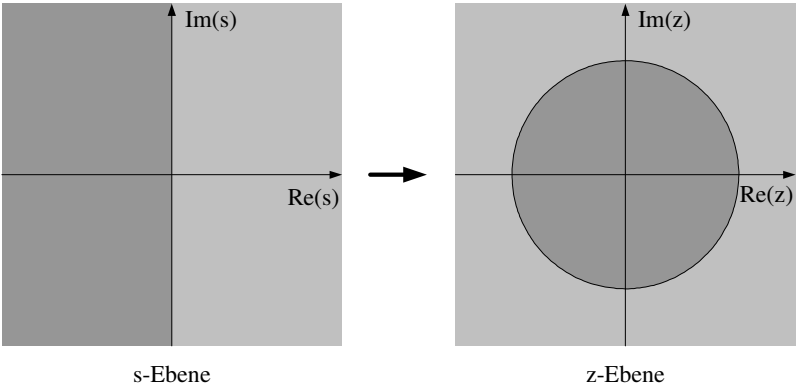


Abb. 1.25. Ebenentransformation

nur Polstellen in der linken offenen s -Halbebene aufweist. Entsprechendes gilt für die z -Ebene im zeitdiskreten Fall. Aufgrund der zuvor beschriebenen Ebenentransformation bildet hier das innere des Einheitskreises den stabilen Bereich.

Wegen der Periodizität von $z = r \cdot e^{j\omega T}$ ist die Abbildung aus dem s -Bereich in die z -Ebene aber nur in Streifen von jeweils $\omega T = 2\pi$ Breite eindeutig, die inverse z -Transformation demzufolge im allgemeinen mehrdeutig. Weiterhin soll nicht unerwähnt bleiben, daß es neben der hier angegebenen zweiseitigen z -Transformation auch eine einseitige Form gibt, bei welcher der Summationsindex k von $0.. \infty$ läuft. Offensichtlich genügt diese Variante zur Transformation kausaler Signale, bei denen die Werte für negative Zeitpunkte ($k < 0$) sämtlich verschwinden.

Tabelle 1.6. Ausgewählte Eigenschaften der z -Transformation

	Zeitbereich	z-Bereich
Linearität	$a \cdot x(k) + b \cdot g(k)$	$a \cdot X(z) + b \cdot G(z)$
Zeitverschiebung	$x(k - n)$	$z^{-n} \cdot X(z)$
Zeitverschiebung (2)	$x(k + n)$	$z^n \cdot X(z) - \sum_{i=0}^{n-1} x(i) \cdot z^{n-i}$
Differentiation	$k \cdot x(k)$	$-z \cdot \frac{dX(z)}{dz}$
Zeitumkehr	$x(-k)$	$X(z^{-1})$
Faltung	$x(k) * g(k) = \sum_{i=0}^k x(i) \cdot g(k-i)$	$X(z) \cdot G(z)$

Die für uns wichtigste Anwendung der z -Transformation liegt natürlich in der Beschreibung digitaler Abtastsysteme, also auch digitaler Filter. Ihr Verhalten im Zeitbereich wird durch Differenzgleichungen charakterisiert. Nach einer Transformation in den z -Bereich erhält man im allgemeinen gebrochene rationale Systemfunktionen, die uns helfen das Frequenzverhalten anschaulich zu verstehen. Auf diesen Punkt wird in den nächsten Abschnitten noch näher eingegangen. An dieser Stelle soll es genügen einige wichtige Eigenschaften der z -Transformation aufzuzeigen.

1.11.3 Die Übertragungsfunktion und der Frequenzgang

Physikalisch betrachtet, beschreibt die Übertragungsfunktion die Antwort eines Systems bei verschiedenen Eingangssignalen, oder genauer ausgedrückt, aufgrund unterschiedlicher (komplexer) Eingangsfrequenzen. In der Meß- und Regelungstechnik wird dieser Umstand häufig genutzt, um die dynamischen Eigenschaften eines unbekannten Systems meßtechnisch zu analysieren (siehe hierzu auch Abschn. 1.12).

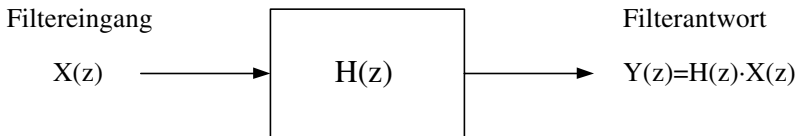


Abb. 1.26. Übertragungsfunktion in Frequenzbereich

Im Frequenzbereich sind das Eingangssignal $X(z)$, die Übertragungsfunktion $H(z)$ und die resultierende Systemantwort $Y(z)$ multiplikativ miteinander verknüpft. Aus diesem Grund ist der Rückschluß auf $H(z)$ durch Umformung leicht möglich.

$$H(z) = \frac{Y(z)}{X(z)} \quad (1.31)$$

Prinzipiell kann aufgrund dieser Gleichung das Übertragungsverhalten durch Messung am Objekt direkt bestimmt werden. In der Praxis bewährt hat sich die Bestimmung des *Frequenzganges* $H(f)$, ein Sonderfall der komplexen Übertragungsfunktion. Um $H(f)$ zu messen, speist man am Eingang ein sinusförmiges Testsignal ein und erhöht dessen Frequenz kontinuierlich. Hat man es mit einem linearen System zu tun, dann wird die Frequenz des Ausgangssignals der Eingangsfrequenz folgen, lediglich die Amplitude und die Phase ändern sich. Diese typische Charakteristik stellt

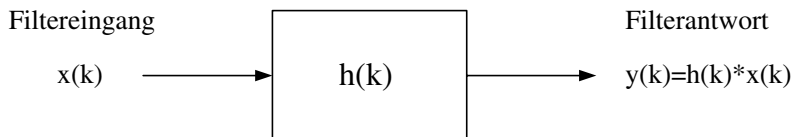


Abb. 1.27. Übertragungsfunktion im Zeitbereich

dann den Frequenzgang des untersuchten Systems dar, spiegelt also seine dynamischen Eigenschaften wider. Aber natürlich können digitale Systeme nicht nur im Frequenzbereich, sondern auch im Zeitbereich beschrieben werden.

Leider sind manche Dinge im Zeitbereich ein wenig komplizierter als im Frequenzbereich. Das elementarste Abtastsignal ist der schon bekannte Einheitsimpuls. Ihn kann man sich als einen unendlichen Datenstrom vorstellen, bei dem genau ein Abtastwert den Wert Eins hat und alle übrigen Werte Null sind. Hat diese spezielle Abtastfolge das System vollständig durchlaufen, dann erhält man an seinem Ausgang die Einheitsimpulsantwort $h(k)$, manchmal auch *Gewichtsfunktion* genannt. Alle Abtastsignale lassen sich nun als Linearkombination gewichteter und verschobener Einheitsimpulse darstellen.

$$x(k) = \sum_{i=-\infty}^{+\infty} x(i) \cdot \delta(k-i) \quad (1.32)$$

Durchläuft eine solche Impulsfolge unser lineares System, erhalten wir am Ausgang ein Signal, welches sich aus der Überlagerung aller entsprechend gewichteten und verschobenen Impulsantworten ergibt. Mathematisch stellt dies eine *Faltungssumme* dar.

$$y(k) = \sum_{i=-\infty}^{+\infty} x(i) \cdot h(k-i) \quad (1.33)$$

Die Kenntnis von $h(k)$ reicht also aus, um die Systemantwort auf beliebige Eingangssignale zu bestimmen. Somit beschreibt die Impulsantwort ein lineares System ebenfalls vollständig. Allerdings sind hier $x(k)$, $h(k)$ und $y(k)$ durch eine Faltung miteinander verknüpft, weshalb im Zeitbereich ein so einfacher Zusammenhang wie Gln. (1.31) leider nicht existiert.

Da die Übertragungsfunktion $H(z)$ die Frequenztransformierte der Einheitsimpulsantwort $h(k)$ ist, läßt sich das Frequenzverhalten eines digitalen Filters mittels z -Transformation auch direkt aus seiner Impulsantwort berechnen. Hier endlich schließt sich der Kreis. Mit der folgenden Gleichung wird die enge Beziehung zwischen Zeit- und Frequenzbereich deutlich, die

untrennbar miteinander verbunden sind. Sie bildet daher einen fundamentalen Grundsatz der digitalen Signalverarbeitung.

$$H(z) = \sum_{k=-\infty}^{+\infty} h(k) \cdot z^{-k} \quad (1.34)$$

Den Frequenzgang $H(f)$ des Filters als Sonderfall der Übertragungsfunktion $H(z)$ erhält man schließlich mit $z = e^{j2\pi f T_a}$ durch Variation der Frequenz f . T_a stellt die gewählte Abtastzeit dar, für die das Filter eingesetzt werden soll. Der Frequenzgang durchläuft alle Werte der Übertragungsfunktion entlang des Einheitskreises in der z -Ebene und wird wegen der besseren Anschauung in Betragsgang und Phasengang zerlegt.

$$H(f) = H(z = e^{j2\pi f T_a}) \quad (1.35)$$

$$H(f) = |H(f)| \cdot e^{j\varphi(f)} \quad (1.36)$$

$$|H(f)| = \sqrt{\Re^2(H(f)) + \Im^2(H(f))} \quad (1.37)$$

$$\varphi(f) = \arctan\left(\frac{\Im(H(f))}{\Re(H(f))}\right) \quad (1.38)$$

Leider ist die Definition des Phasenganges $\varphi(f)$ in der Literatur nicht ganz einheitlich. Während in der Regelungstechnik meist die vorliegende Form verwendet wird, ist es in der Nachrichtentechnik üblich, den Phasengang zusätzlich mit einem negativen Vorzeichen zu versehen. Da die digitale Signalverarbeitung gleichermaßen in beiden Disziplinen verwurzelt ist, sind in den Lehrbüchern auch beide Definitionen vorzufinden. Insbesondere in der deutschsprachigen Literatur wird für die Übertragungsfunktion auch häufig $G(z)$ statt $H(z)$ verwendet, entsprechend $g(k)$ anstelle von $h(k)$. Hiervon sollte man sich aber nicht beirren lassen.

1.11.4 Mittelwertfilter

Das wohl am häufigsten benutzte digitale Filter ist das *Moving-Average-Filter* (MA-Filter), welches einfach den Mittelwert aus einer Anzahl von Meßwerten berechnet. Es ist so simpel, daß man bei seiner Verwendung fast ein schlechtes Gewissen hat, wegen der scheinbar mangelnden Wissenschaftlichkeit eines solchen Ansatzes. Hierzu besteht jedoch kein Grund, ganz im Gegenteil. Wegen seiner einfachen Handhabung und verständlichen Wirkungsweise ist es leicht zu implementieren und eignet sich

damit besonders zur Reduzierung von zufälligen Störungen, wie sie bei der Meßwerterfassung eigentlich immer auftreten.

$$y(k) = \frac{1}{M} \sum_{i=0}^{M-1} x(k-i) \quad (1.39)$$

Hierbei sind $x(k)$ die Abtastwerte des Eingangssignals, M die Anzahl der einbezogenen Abtastwerte und $y(k)$ das gefilterte Ausgangssignal. Bei Online-Anwendungen kann für eine Filterung nur auf den aktuellen Meßwert $x(k)$ und zeitlich zurückliegende Eingangswerte zugegriffen werden. Ein solches Filter ist *kausal*. Liegen die Eingangsdaten zum Zeitpunkt der Verarbeitung schon vor, zum Beispiel abgespeichert in einer Meßwertdatei, dann ist grundsätzlich auch die Verwendung nichtkausaler Filter denkbar. Bei ihnen werden zur Berechnung des aktuellen Ausgangswertes neben dem aktuellen Abtastwert auch vergangene und zukünftige Werte einbezogen, meist symmetrisch angeordnet.

$$y(k) = \frac{x(k-2) + x(k-1) + x(k) + x(k+1) + x(k+2)}{5}$$

In der hier vorgestellten nichtrekursiven Form hängt der Ausgangswert ausschließlich von den Eingangswerten ab. Möchte man den Mittelwert aus einer Anzahl von Meßwerten bestimmen, muß man warten, bis tatsächlich genügend Werte aus der Messung vorliegen. Dabei wächst die Antwortzeit des Filters mit der Breite seines Datenfensters, d. h. mit der Anzahl der einbezogenen Meßwerte. Auch wenn für eine Mittelwertbildung nach Gln. (1.39) sehr viele Meßwerte herangezogen werden, sind es doch immer nur endlich (also abzählbar) viele, weshalb auch die Filterantwort immer nach einer endlichen Zeit vorliegt. Da sie in jedem Fall endlich ist, gehören nichtrekursive Filter zur Klasse der *FIR-Filter* (Finite Impulse Response). Bekanntermaßen gehen beim Mittelwertfilter alle Abtastwerte mit dem gleichen Gewicht $1/M$ ein. Die identischen Filterkoeffizienten bilden im Zeitbereich ein Rechteckfenster, welches die Eigenschaften und das Verhalten des Filters im Frequenzbereich bestimmt.

Neben den nichtrekursiven FIR-Filtern, bei denen die Filterantwort zu jedem Zeitpunkt ausschließlich von den anliegenden Eingangswerten abhängt, gibt es auch *rekursive Filter*. Aufgrund ihrer besonderen Struktur gehen bei diesem Filtertyp die Ausgangswerte zurückliegender Zeitpunkte

ebenfalls in die aktuelle Filterantwort ein. Wie die folgende Beispielrechnung belegt, kann man ein nichtrekursives Mittelwertfilter ohne großen Aufwand in eine rekursive Form bringen.

$$y(58) = \frac{x(56) + x(57) + x(58) + x(59) + x(60)}{5}$$

$$y(59) = \frac{x(57) + x(58) + x(59) + x(60) + x(61)}{5}$$

$$y(58) - y(59) = \frac{x(56) - x(61)}{5}$$

$$y(59) = y(58) + \frac{x(61) - x(56)}{5}$$

Wie sich zeigt, hängt der Ausgangswert $y(59)$ von den Eingangswerten $x(56)$, $x(61)$ und dem alten Ausgangswert $y(58)$ ab. Man kann sich nun leicht überlegen, daß es in einem nächsten Schritt durchaus möglich wäre, den Ausgangswert $y(58)$ ebenfalls rekursiv durch seinen Vorgänger $y(57)$ zu ersetzen, mit dem man dann wiederum so verfahren könnte. Da somit beliebig viele alte Ausgangswerte in die aktuelle Filterantwort eingehen, ist folglich auch die Filterantwort selbst beliebig lang. Rekursive Filter zählen daher zur Klasse der *IIR-Filter* (Infinite Impulse Response). Schaut man sich das obige Beispiel an, wird schnell klar, daß ein wesentlicher Vorteil rekursiver Filter in der schnelleren Berechenbarkeit durch einen Computer zu sehen ist, da in dieser kompakten Form erheblich weniger Adreßaufrufe und Rechenoperationen notwendig sind.

1.11.5 FIR-Filter

FIR-Filter bilden im Zeitbereich nichtrekursive Differenzengleichungen mit konstanten Koeffizienten. Als Beispiel dient uns erneut die schon bekannte Mittelwertbildung aus fünf Meßwerten.

$$y(k) = \frac{1}{5} \cdot (x(k) + x(k-1) + x(k-2) + x(k-3) + x(k-4))$$

Mit Hilfe der z -Transformation ist die Transformation dieser Gleichung und ihre Betrachtung im Frequenzbereich möglich.

$$Y(z) = \frac{1}{5} \cdot (X(z) + z^{-1} \cdot X(z) + z^{-2} \cdot X(z) + z^{-3} \cdot X(z) + z^{-4} \cdot X(z))$$

$$Y(z) = \frac{1}{5} \cdot X(z) \cdot (1 + z^{-1} + z^{-2} + z^{-3} + z^{-4})$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{5} \cdot (1 + z^{-1} + z^{-2} + z^{-3} + z^{-4})$$

Als Ergebnis unserer Berechnung erhalten wir die Übertragungsfunktion $H(z)$. Genau wie die Differenzengleichung im Zeitbereich, charakterisiert die Übertragungsfunktion im Frequenzbereich das Filters vollständig. Bei linearen Systemen bildet die Übertragungsfunktion eine sehr anschauliche Eingangs-/Ausgangsbeziehung. Eine solche Interpretation ist jedoch nur im Frequenzbereich möglich, was eindeutig für die Verwendung der z -Transformation spricht. Differenzengleichung und Übertragungsfunktion lassen sich ganz allgemein formulieren:

$$y(k) = a_0 \cdot x(k) + a_1 \cdot x(k-1) + a_2 \cdot x(k-2) + \dots + a_N \cdot x(k-N) \quad (1.40)$$

$$H(z) = \sum_{i=0}^N a_i \cdot z^{-i} = a_0 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2} + \dots + a_N \cdot z^{-N} \quad (1.41)$$

Auch wenn es nicht auf den ersten Blick erkennbar ist, stellt der letzte Ausdruck eine gebrochen rationale Funktion der komplexen Variable z dar. Wir formen Gln. (1.41) um, indem wir mit z^N erweitern.

$$H(z) = \frac{a_0 \cdot z^N + a_1 \cdot z^{N-1} + a_2 \cdot z^{N-2} + \dots + a_N}{z^N}$$

Man erkennt jetzt den N -fachen Pol der Übertragungsfunktion bei $z=0$, der aber auf die Stabilität des Filters keinen Einfluß hat, da er innerhalb des Einheitskreises liegt (mit $z = e^{sT}$ entspricht das der komplexen Frequenz $s = -\infty$). Dieser Umstand ist typisch für FIR-Filter, die immer stabil sind, da ihre Polstellen ausschließlich im Koordinatenursprung der z -Ebene liegen.

Möchte man die Übertragungsfunktion bzw. den Frequenzgang eines FIR-Filters bestimmen, dann ist das nach Gln. (1.34) mit Hilfe der z -Transformation auch unmittelbar aus der Impulsantwort möglich. Hierzu ist es aber notwendig, die Impulsantwort des Filters zu kennen. Sollte das nicht der Fall sein, kommt man nicht umhin sie zu bestimmen. Wie das funktioniert ist klar. Zunächst ist am Filtereingang ein Einheitsimpuls anzulegen, der das Filter vollständig zu durchlaufen hat. Am Filterausgang erhält man dann die gesuchte Impulsantwort. Glücklicherweise ist das bei FIR-Filtern ganz besonders einfach.

Als Ergebnis erhält man am Filterausgang gerade die einzelnen Filterkoeffizienten a_i selbst. Das durch sie gebildete Signalfenster stellt also die

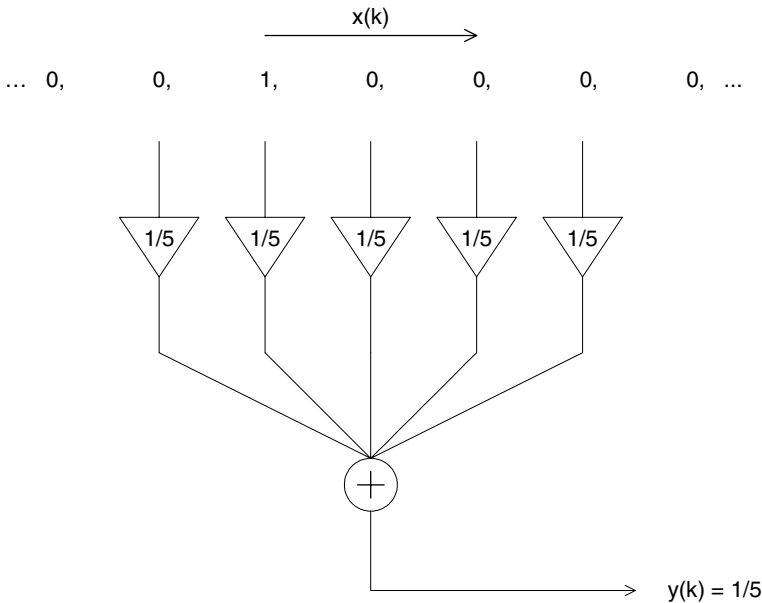


Abb. 1.28. Nichtrekursives Mittelwertfilter mit Einheitsimpuls am Eingang

Einheitsimpulsantwort des FIR-Filters im Zeitbereich dar. Komplizierte Berechnungen sind somit überflüssig.

Entwurf und Analyse von FIR-Filtern

Frequenzselektive Filter werden eingesetzt, um bestimmte Frequenzanteile aus einem Signal herauszufiltern. Ganz gleichgültig, welcher Filtertyp in einer konkreten Anwendung zum Einsatz kommen soll, in jedem Fall ist es für den Filterentwurf zwingend notwendig, eine möglichst exakte Vorstellung von dem gewünschten Frequenzverhalten des Filters zu haben. Nur so lassen sich, entsprechend der Aufgabenstellung, die Anforderungen an das Filter genau spezifizieren. Somit erfolgt die Wahl der Filterkoeffizienten immer problemabhängig. Unter dem Entwurf eines Filters wollen wir daher die Bestimmung geeigneter Koeffizienten verstehen. Die Theorie des Filterentwurfs ist ein sehr weitläufiges Thema und kann an dieser Stelle nicht einmal annähernd vollständig behandelt werden. Aus diesem Grund stehen auch eine Reihe kommerzieller Entwurfsprogramme zur Verfügung, die den Entwickler bei seiner Arbeit unterstützen. Die hier vorgestellten Methoden ermöglichen ein tieferes Verständnis für die Probleme dieser Thematik und versetzen den Leser in die Lage, die Funktionsweise softwaregestützter Entwurfsverfahren besser zu verstehen.

Die Fenstermethode

Beim Entwurf von FIR-Filtern ist ein Verfahren gebräuchlich, für das die bisher erarbeiteten Grundlagen eine ausgezeichnete Basis bieten. Die nachfolgend vorgestellte *Fenstermethode* zeichnet sich besonders durch ihr unkompliziertes und intuitives Vorgehen aus. Man muß sich lediglich in Erinnerung rufen, daß die inverse Fouriertransformierte des Frequenzganges $H(f)$ die Impulsantwort $h(k)$ des Filters ergibt. Diese ist aber bei FIR-Filtern gerade mit den Filterkoeffizienten identisch. Somit beschränkt sich der Entwurf nach der Fenstermethode darauf, den gewünschten Filterfrequenzgang vorzugeben und die gesuchten Filterkoeffizienten zu berechnen, beispielsweise mit der schon bekannten IDFT. Wenn doch nur alles im Leben so einfach wäre, oder doch nicht? Die Anforderungen an ein Filter sind schnell formuliert. Neben der angestrebten Grenzfrequenz, stehen ein glatter Frequenzverlauf im Durchlaßbereich, ein möglichst steiler Übergang und schließlich eine gute Dämpfung im Sperrbereich im Vordergrund. Gleichzeitig strebt man einen kleinen Filtergrad an, um die Reaktionszeit des Filters und den Berechnungsaufwand bei der Faltung mit einem Signal niedrig zu halten. Alle diese typischen Forderungen gemeinsam unter einen Hut zu bekommen ist aber schwieriger als es zunächst den Anschein hat. Tatsächlich widersprechen sie sich sogar zum Teil, wie wir am Beispiel eines Tiefpaßfilters mit $f_g = 1/16 f_a$ sehen werden. Das angestrebte Übertragungsverhalten ist in Abb. 1.29 dargestellt und dient als Ausgangspunkt für den weiteren Entwurf.

Wie wir schon im Abschnitt über Pulse und Pulsfolgen gesehen haben, berechnet die DFT (und damit natürlich auch die FFT) immer ein bei der halben Abtastfrequenz gespiegeltes Spektrum. Gibt man nun umgekehrt einen Frequenzgang vor, um mit Hilfe der IDFT die Impulsantwort des Filters zu berechnen, darf man diese periodische Fortsetzung im Frequenzbereich nicht vergessen. Wird ein solcher idealer Wunschfrequenzgang

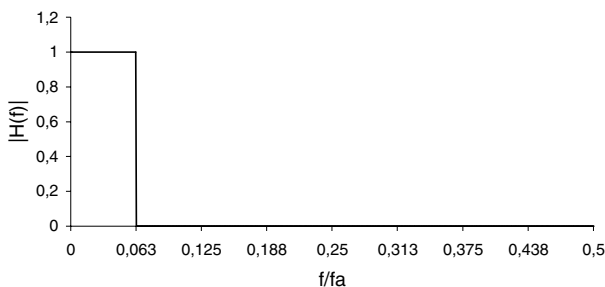


Abb. 1.29. Wunschfrequenzgang eines Tiefpaßfilters mit $f_g = 1/16 f_a$

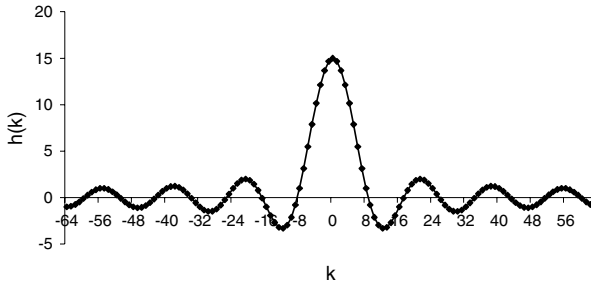


Abb. 1.30. Impulsantwort des Tiefpaßfilters

gewählt, erhält man als Ergebnis der IDFT die schon bekannte Spaltfunktion, deren Abtastwerte $h(k)$ schließlich die gesuchten Filterkoeffizienten bilden.

Zumindest theoretisch stimmt das so. Nimmt man aber einen auf diese Weise erzeugten Satz von Filterkoeffizienten und schaut auf die hieraus resultierende Übertragungsfunktion (wie das geht, sehen wir im nächsten Abschnitt), dann erlebt man eine unangenehme Überraschung.

Der tatsächliche Verlauf des Amplitudenfrequenzganges weicht ganz erheblich von der angestrebten Idealform ab. Ein Blick zurück auf die berechnete Impulsantwort unseres Tiefpaßfilters in Abb. 1.30 legt es nahe, als Ursache die Verwendung von nur endlich vielen Filterkoeffizienten aus der doch eigentlich nach beiden Seiten unendlich ausgedehnten Spaltfunktion zu vermuten. Eine solche Begrenzung der Impulsantwort kommt der Multiplikation mit einem Rechteckfenster gleich und ist für reelle Anwendungen durchaus sinnvoll, schließlich sind in der Praxis Filter unendlicher Länge nicht zu gebrauchen. Unvermeidlich bringt das Abschneiden der Filterkoeffizienten dann aber auch eine ganze Reihe unerwünschter Nebenwirkungen im Frequenzbereich mit sich.

- Es entsteht im Durchlaßbereich eine deutliche Welligkeit, deren Amplitude zur Grenzfrequenz hin sogar noch zunimmt (Gibbsches Phänomen).
- Im Sperrbereich des Filters sind mehrere Nebenmaxima erkennbar, die nur langsam mit zunehmender Frequenz abklingen.
- Der Übergangsbereich ist erheblich aufgeweitet, die Filterflanke demzufolge keineswegs ideal steil.

Alle diese Nachteile könnten durch eine Vergrößerung der Filterlänge grundsätzlich behoben werden, wenn diese denn beliebig groß wählbar wäre. Insbesondere die Breite des Übergangsgebietes ist unmittelbar mit der Filterlänge verknüpft und ließe sich auf diese Weise leicht dem angestrebten

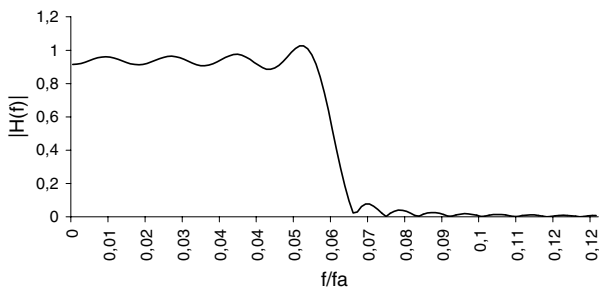


Abb. 1.31. Tatsächlicher Frequenzgang des Filters (geänderte Skalierung!)

Ideal annähern. Mit zunehmender Filterlänge wächst dann allerdings auch die Anzahl der erforderlichen Rechenoperationen, was mit einem entsprechend hohem Rechenzeitbedarf verbunden ist, egal ob eine Softwareimplementierung oder eine Hardwarerealisierung des Filter vorliegt.

Der bisher erzielte Filterfrequenzgang in Abb. 1.31 ist jedoch für viele Anwendungen nicht akzeptabel, weshalb wir unseren prinzipiell richtigen Ansatz noch ein wenig verfeinern müssen. Schon im Abschn. 1.8 haben wir festgestellt, daß die Verwendung von Rechteckfenstern keineswegs immer zu optimalen Ergebnissen führt. Es konnte gezeigt werden, daß sich die in bestimmten Anwendungsfällen auftretenden spektralen Unschärfen durch die Wahl geeigneterer Fenstertypen deutlich mildern lassen. Das trifft auch auf den von uns angestrebten Filterfrequenzgang zu. Bevor wir jedoch diese Zusammenhänge näher untersuchen, ist es zweckmäßig einige der bekanntesten Fenstertypen hier kurz einmal vorzustellen. In den folgenden Beispielen ist M die Anzahl der Abtastwerte je Signalperiode, also die Fensterlänge.

Hanning:

$$w(k) = 0,5 - 0,5 \cdot \cos\left(\frac{2\pi k}{M}\right)$$

Hamming:

$$w(k) = 0,54 - 0,46 \cdot \cos\left(\frac{2\pi k}{M}\right)$$

Blackman:

$$w(k) = 0,42 - 0,5 \cdot \cos\left(\frac{2\pi k}{M}\right) + 0,08 \cdot \cos\left(\frac{4\pi k}{M}\right)$$

Blackman (exakt):

$$w(k) = 0,42659071 - 0,49656062 \cdot \cos\left(\frac{2\pi k}{M}\right) + 0,07684867 \cdot \cos\left(\frac{4\pi k}{M}\right)$$

Blackman-Harris:

$$w(k) = 0,42323 - 0,49755 \cdot \cos\left(\frac{2\pi k}{M}\right) + 0,07922 \cdot \cos\left(\frac{4\pi k}{M}\right)$$

Flat Top:

$$w(k) = 0,2810639 - 0,5208972 \cdot \cos\left(\frac{2\pi k}{M}\right) + 0,1980399 \cdot \cos\left(\frac{4\pi k}{M}\right)$$

Bartlett:

$$w(k) = 1 - \left| \frac{k - M/2}{M/2} \right|$$

Welch:

$$w(k) = 1 - \left| \frac{k - M/2}{M/2} \right|^2$$

Mit dem Aufkommen der digitalen Rechentechnik in der Signalverarbeitung, etwa ab 1950, wurden die geschilderten Probleme bei der Benutzung von Rechteckfenstern sehr schnell erkannt. Die Cosinus-Funktion bietet sich in diesem Fall als naheliegende Lösung der durch das abrupte

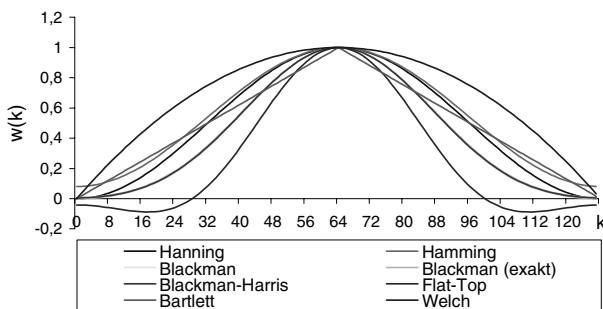


Abb. 1.32. Hüllkurven der verschiedenen Fensterfunktionen

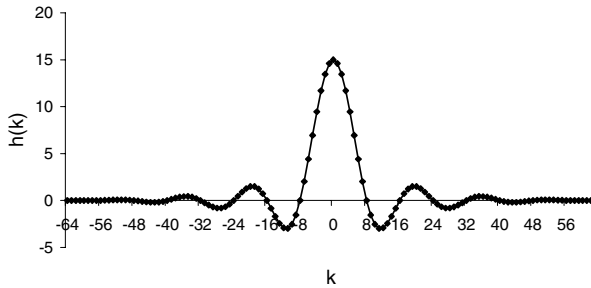


Abb. 1.33. Impulsantwort nach Multiplikation mit einem Hanning-Fenster

Abschneiden verursachten Abweichungen vom Ideal an. Tatsächlich ist sie der Urahn der allermeisten Fenstertypen. Hieraus sind dann im Laufe der Zeit zahlreiche Varianten entstanden, die häufig nach ihrem Entwickler benannt wurden. Trotz der Vielfalt sei aber darauf hingewiesen, daß die häufigste Fensterfunktion immer noch das Rechteckfenster selbst ist.

Alle aufgeführten Fenster liegen symmetrisch um einen mittleren Abtastzeitpunkt. Mit ihnen gelingt es, die Filterkoeffizienten zum Rand hin kontrolliert gegen einen vernachlässigbaren Wert zu zwingen. Hierfür ist es erforderlich die in Abb. 1.30 dargestellten Rohwerte punktweise mit $w(k)$ zu multiplizieren.

Genau genommen kann der Filterentwurf bis zu diesem Punkt sogar noch verkürzt werden. Das unsere ideale Vorgabe im Frequenzbereich als Impulsantwort eine Spaltfunktion besitzt, ist ja ohnehin klar. Warum sollte es dann noch notwendig sein, dieses doch eigentlich schon bekannte Ergebnis über eine IDFT zu erzeugen? Das kann man auch schneller und einfacher haben, Zeit ist schließlich Geld. Die Tatsache, daß die angestrebte Grenzfrequenz f_g und die Zahl der Abtastwerte M den Verlauf von $h(k)$ eindeutig festlegen, wollen wir für unsere Zwecke nutzen.

$$h(k) = K \cdot \frac{\sin(2\pi f_g (k - M/2))}{(2\pi f_g (k - M/2))} \cdot w(k) \quad (1.42)$$

Nimmt man für $w(k)$ eine der zuvor aufgelisteten Fensterfunktionen, dann kann die gefensterte Impulsantwort mit Hilfe dieser Gleichung direkt berechnet werden. Die Konstante K dient zur Einstellung der gewünschten Gleichspannungsverstärkung $|H(0)|$. Bei der Implementierung ist zu beachten, daß an der Stelle $k = M/2$ ein „Division durch Null“-Fehler auftritt, wenn k über alle Abtastwerte läuft. Natürlich wollen wir das vermeiden, die Addition einer kleinen Konstanten (zum Beispiel 0,00001) im Argument des Sinus und im Nenner wäre eine Möglichkeit hierzu. Unabhängig

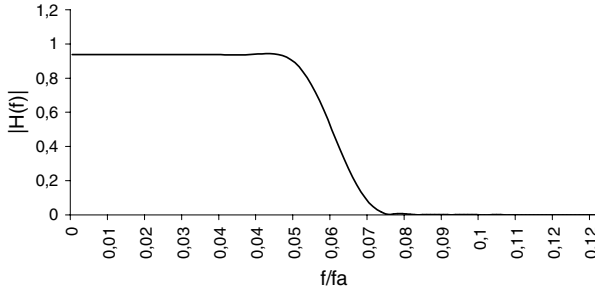


Abb. 1.34. Amplitudengang nach Hanning-Fensterung der Impulsantwort

davon, ob wir uns für die IDFT oder Gln. (1.42) entscheiden, zeigt die zusätzliche Fensterung der Impulsantwort eine deutliche Verbesserung im resultierenden Frequenzgang.

Auf den ersten Blick ist zu erkennen, daß die störende Welligkeit im Durchlaßbereich weitgehend verschwunden ist und auch die Sperrdämpfung sich deutlich erhöht hat. Beides hängt natürlich vom konkret verwendeten Fenstertyp ab. Interessanterweise ist die Verstärkung unseres Filters, entgegen der Vorgabe, ein wenig kleiner als Eins. Das war übrigens auch in Abb. 1.31 schon so, wenngleich es dort wegen der ausgeprägten Welligkeit nicht sofort auffällt. Ursache ist erneut die nur endliche Anzahl von Filterkoeffizienten aus der doch eigentlich unendlich ausgedehnten Spaltfunktion. Hier macht sich die Vernachlässigung kleiner und scheinbar unbedeutender Amplituden bemerkbar. In den meisten Fällen ist jedoch eine Gleichspannungsverstärkung von genau Eins erwünscht, weshalb wir die Koeffizienten noch entsprechend normieren müssen. Dazu dividiert man jeden einzelnen Filterkoeffizienten durch die Summe aller Koeffizienten. Weiterhin besitzt unser Filter noch nicht die gewünschte Grenzfrequenz. Für einen Tiefpaß ist sie definiert als

$$H_{TP}(f_g) = \frac{1}{\sqrt{2}} \cdot H_{TP}(0) \quad (1.43)$$

Abhängig von der Fensterfunktion ist die tatsächlich resultierende Grenzfrequenz ein wenig kleiner als die Vorgabe, was es notwendig macht, den angestrebten Wert zum Ausgleich entsprechend anzuheben. Somit ist der gesamte Entwurfsprozeß mit der modifizierten Grenzfrequenz f_g zu wiederholen. Dies führt zu einem Iterationsvorgang, der unter Umständen mehrmals durchlaufen werden muß. Auch wenn das zunächst kompliziert erscheint, kommt man in der praktischen Anwendung recht schnell zu einem akzeptablen Ergebnis.

Betrachtet man das bisher gezeigte Vorgehen, kommt zwangsläufig die Frage auf, für welche der zahlreichen Fensterfunktion man sich denn nun entscheiden soll. Leider kann dies nicht eindeutig beantwortet werden, sondern hängt von den jeweiligen Anforderungen ab. Wie nicht anders zu erwarten, beeinflussen die verschiedenen Fenster den resultierenden Frequenzgang in unterschiedlicher Weise. Die Wahl einer geeigneten Funktion ist aus diesem Grund immer nur anwendungsspezifisch möglich. In den meisten Fällen sind jedoch das Hanning-, Hamming- oder das Blackman-Fenster eine gute Wahl. Sie haben sich zu Recht als echte Arbeitspferde in der digitalen Signalverarbeitung etabliert.

Nun gut, aber was ist, wenn kein Tiefpaßfilter benötigt wird, sondern ein Hochpaß, ein Bandpaß oder eine Bandsperre? Gibt es für diese Fälle auch eine Formel, mit deren Hilfe man die Filterkoeffizienten ohne großen Aufwand berechnen kann, oder bleibt hier nur die IDFT? Tatsächlich gibt es noch einen dritten Weg, nicht ganz so einfach wie die Benutzung von Gln. (1.42), viel komplizierter aber auch nicht. Dieser neue Weg nutzt den Umstand, daß sich aus einem Tiefpaß auch die anderen Filtertypen entwickeln lassen. Alles was wir dafür noch benötigen, ist ein *Allpaßfilter*, also ein Filter, dessen Durchlaßbereich von $f=0$ bis $f=\infty$ reicht, d. h. es gilt $|H_{AP}(f)|=1$ für sämtliche Frequenzen. Schauen wir noch einmal auf Abb. 1.23 und führen uns dabei das Übertragungsverhalten eines Allpaßfilters vor Augen, dann kann man leicht nachvollziehen, daß sich z. B. ein Hochpaß aus der Differenz von Allpaß und Tiefpaß ergibt. Mit Allpaß, Tiefpaß und Hochpaß lassen sich dann durch Addition bzw. Subtraktion auch Bandpaß und Bandsperre konstruieren. Für normierte Frequenzgänge, also solche mit einem Verstärkungsfaktor=1, gelten dann die folgenden Zusammenhänge.

Hochpaß:

$$\begin{aligned} H_{HP}(f) &= H_{AP}(f) - H_{TP}(f) \\ H_{HP}(f) &= 1 - H_{TP}(f) \end{aligned} \quad (1.44)$$

Bandpaß:

$$\begin{aligned} H_{BP}(f) &= H_{TP}(f) + H_{HP}(f) - H_{AP}(f) \\ H_{BP}(f) &= H_{TP}(f) + H_{HP}(f) - 1 \end{aligned} \quad (1.45)$$

mit

$$f_{g\ TP} > f_{g\ HP}$$

Bandsperre:

$$H_{BS}(f) = H_{AP}(f) - H_{BP}(f)$$

$$H_{BS}(f) = 1 - H_{BP}(f) \quad (1.46)$$

Die Gleichungen dienen eigentlich nur dem Zweck, die Richtigkeit unseres Entwurfsverfahrens zu veranschaulichen. Wirklich anwenderfreundlich wird es durch die Tatsache, daß die exakt gleichen Beziehungen auch für die Filterkoeffizienten $h(k)$ im Zeitbereich gelten, d.h. wir müssen diese nur in geeigneter Weise addieren und subtrahieren, um den jeweils gewünschten Filtertyp zu erhalten. Bleibt eigentlich nur noch zu klären, wie denn die Koeffizienten eines Allpaßfilters aussehen könnten. Hier hilft erneut ein Blick auf Tabelle 1.4 weiter. Ein Allpaß hat genau einen Filterkoeffizienten, da wir normierte Filter betrachten, hat dieser den Wert Eins. Damit der Entwurf funktioniert, ist der AP-Koeffizient mit dem jeweils zentralen Koeffizienten unserer TP/HP/BP/BS-Filter zu addieren bzw. zu subtrahieren.

```
class Filter
{
    const int Abtastwerte = 128;
    const double PI      = 3.141592653589793;

    // Filterkoeffizienten
    public double[] x = new double[Abtastwerte];
    public int filterlaenge;

    ...
    public void TPKoeffizienten(double fg)
    {
        double sum = 0;

        for (int k=0; k<filterlaenge; k++)
        {
            // Spaltfunktion
            x[k] = Math.Sin(1.0E-10 +
                           2*PI*fg*(k-filterlaenge/2))
                /
                (1.0E-10 + 2*PI*fg*(k-filterlaenge/2));

            // Hanning-Fensterung
            x[k] *= 0.5 - 0.5*Math.Cos(2*PI*k/filterlaenge);

            sum += x[k];
        }
    }
}
```

```

    }

    // Normierung der Koeffizienten
    for (int k=0; k<filterlaenge; k++)
    {
        x[k] /= sum;
    }
}
}

```

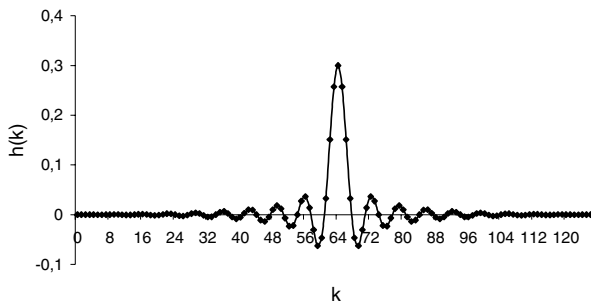


Abb. 1.35. Koeffizienten eines Tiefpaßfilters mit $f_g/f_a = 0,15$

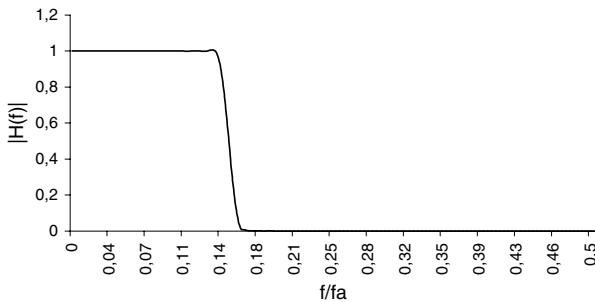


Abb. 1.36. TP-Amplitudengang mit $f_g/f_a = 0,15$

```

class Filter
{
    ...
    public void HPKoeffizienten(double fg)
    {
        // Basis für den Entwurf ist ein Tiefpass
        TPKoeffizienten(fg);

        // Tiefpass invertieren
    }
}

```

```

for (int k=0; k<filterlaenge; k++)
{
    x[k] = -x[k];
}

// Allpass addieren
x[filterlaenge/2] += 1.0;
}
}

```

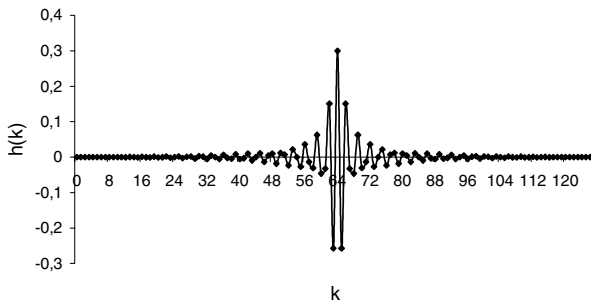


Abb. 1.37. Koeffizienten eines Hochpaßfilters mit $f_g/f_a = 0,35$

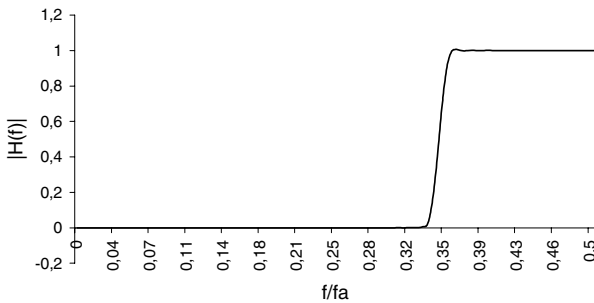


Abb. 1.38. HP-Amplitudengang mit $f_g/f_a = 0,35$

```

class Filter
{
    ...
    public void BPKoeffizienten(double fg1, double fg2)
    {
        // lokale Kopien
        double[] htp = new double[filterlaenge];
        double[] hhp = new double[filterlaenge];
    }
}

```

```

TPKoeffizienten(fg2);
for (int k=0; k<filterlaenge; k++)
{
    htp[k] = x[k];
}

HPKoeffizienten(fg1);
for (int k=0; k<filterlaenge; k++)
{
    hhp[k] = x[k];
}

// Tiefpass und Hochpass addieren
for (int k=0; k<filterlaenge; k++)
{
    x[k] = htp[k] + hhp[k];
}

// Allpass subtrahieren
x[filterlaenge/2] -= 1.0;
}
}

```

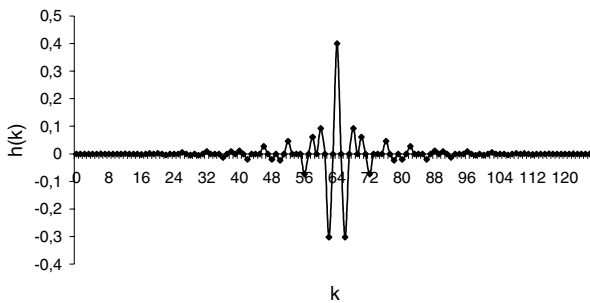


Abb. 1.39. Koeffizienten eines Bandpaßfilters mit $f_{g1}/f_a = 0,15$ und $f_{g2}/f_a = 0,35$

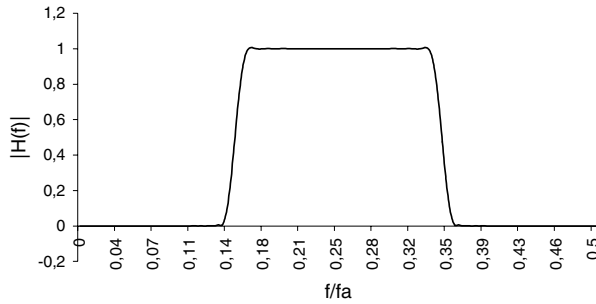


Abb. 1.40. BP-Amplitudengang mit $f_{g1}/f_a = 0,15$ und $f_{g2}/f_a = 0,35$

```
class Filter
{
    ...
    public void BSKoeffizienten(double fg1, double fg2)
    {
        // Basis für den Entwurf ist ein Bandpass
        BPKoeffizienten(fg1, fg2);

        // Bandpass invertieren
        for (int k=0; k<filterlaenge; k++)
        {
            x[k] = -x[k];
        }

        // Allpass addieren
        x[filterlaenge/2] += 1.0;
    }
}
```

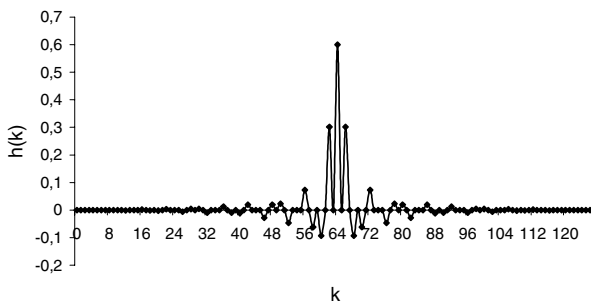


Abb. 1.41. Koeffizienten eines Bandsperrefilters mit $f_{g1}/f_a = 0,15$ und $f_{g2}/f_a = 0,35$

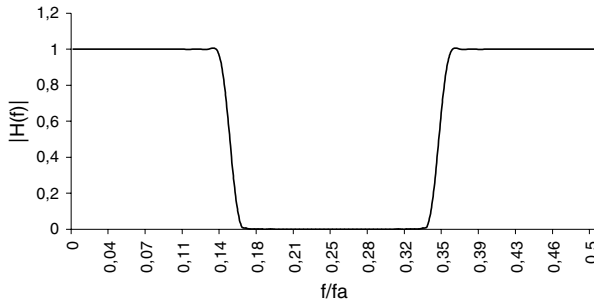


Abb. 1.42. BS-Amplitudengang mit $f_{g1}/f_a = 0,15$ und $f_{g2}/f_a = 0,35$

Von den FIR-Koeffizienten zum Frequenzgang

Bei der Fenstermethode wird ein Frequenzgang vorgegeben, um daraus die gesuchten Filterkoeffizienten zu ermitteln. Besonders für die Filteranalyse ist aber auch der umgekehrte Weg von Nutzen. Hier kommt es darauf an, aus den bekannten Koeffizienten auf das Übertragungsverhalten zu schließen. Denkbar ist aber auch, daß die Koeffizienten zu modifizieren sind, um den Frequenzgang des Filters an veränderte Randbedingungen anzupassen. Wir wollen daher in diesem Abschnitt den unmittelbaren Zusammenhang zwischen Filterkoeffizienten und Frequenzgang näher untersuchen.

Wie schon aus Gln. (1.34) bekannt, ergibt sich die Übertragungsfunktion eines Filters durch z -Transformation der Impulsantwort und kann somit direkt aus den Filterkoeffizienten berechnet werden. Glücklicherweise ist das genauso einfach, wie es auf den ersten Blick erscheint, Computer sei Dank. Die DFT als Sonderfall der z -Transformation ist hierfür hervorragend geeignet und steht uns auch schon als Implementierung zur Verfügung. Von einigen kleinen Änderungen einmal abgesehen, kann auf den alten Quellcode zurückgegriffen werden. Notwendige Anpassungen ergeben sich aus dem Umstand, daß bei der Bestimmung des Übertragungsverhaltens der interessierende Frequenzbereich manchmal kleiner ist als bei der Spektralanalyse und die betrachteten Frequenzen dementsprechend nicht nur ganzzahlige Vielfache der Grundschwingung sind. Auf die bei der DFT vorgenommene Korrektur des Gleichanteils und die Normierung auf die Abtastwerte kann verzichtet werden. Als Testsignal sind dann einfach die Filterkoeffizienten selbst zu nehmen.

```
class Filter
{
    ...
    public void ErzeugeTestsignal_1()
```

```

{
    int M = 10;

    for (int k=0; k<Abtastwerte; k++)
    {
        // Koeffizienten des MW-Filter
        if (k < M)
        {
            x[k] = 1.0/(double)M;
        }
        else
        {
            x[k] = 0;
        }
    }
}
...

public void FilterFrequenzgang(float startfreq,
                               float stopfreq)
{
    float deltaf = (stopfreq-startfreq)/(Abtastwerte-1);
    float freq   = startfreq;
    float real    = 0;
    float imag    = 0;

    // über alle Frequenzen
    for (int i=0; i<Abtastwerte; i++)
    {
        real = 0;
        imag = 0;

        // über alle Filterkoeffizienten
        for (int k=0; k<filterlaenge; k++)
        {
            real += x[k]*Math.Cos(2*PI*freq*k/Abtastwerte);
            imag += x[k]*Math.Sin(2*PI*freq*k/Abtastwerte);
        }

        AS[i] = Math.Sqrt(real*real + imag*imag);
        PS[i] = Math.Atan2(imag, real);

        freq += deltaf;
    }
}
...
}

```

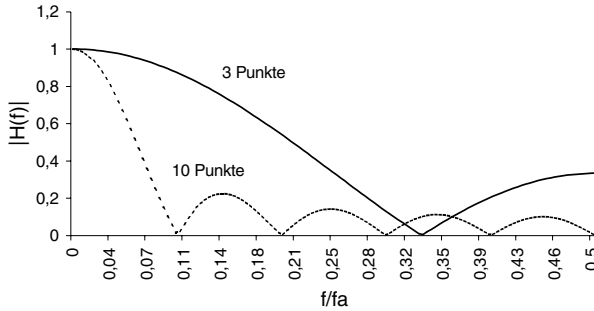


Abb. 1.43. Amplitudenfrequenzgang eines Mittelwertfilters mit 3 und 10 Werten

Verwendet man die Koeffizienten eines Mittelwertfilters, dann zeigt der resultierende Amplitudenfrequenzgang dessen typische Tiefpaßeigenschaft. Wegen der geringen Flankensteilheit im Übergangsbereich und der ausgeprägten Welligkeit im Sperrbereich eignen sich Mittelwertfilter nicht besonders gut zur Trennung verschiedener Signalfrequenzen. Vielmehr werden sie meist zur Reduktion zufälliger Störungen in Meßsignalen eingesetzt. Dabei wächst die Störgrößenunterdrückung des Mittelwertfilters erwartungsgemäß mit der Anzahl der einbezogenen Meßwerte. Je mehr Werte man verwendet, desto zuverlässiger werden die überwiegend hochfrequenten Störungen auch tatsächlich herausgefiltert. Abhängig von der Filterbreite M und der Abtastfrequenz f_a werden bestimmte Frequenzanteile sogar vollständig unterdrückt.

$$\left| H\left(f = \frac{n \cdot f_a}{M}\right) \right| = 0 \text{ für } n \text{ ganzzahlig}$$

Letztlich sind FIR-Filter ein Spezialfall der allgemeineren IIR-Filter. Nur haben bei ihnen sämtliche Koeffizienten des Rückkopplungszweiges den Wert Null. Bezüglich der Entwurfs- und der Analysemöglichkeiten von FIR-Filtern im Frequenzbereich (basierend auf deren Nullstellen in der z -Ebene) sei deshalb auf die nächsten Abschnitte verwiesen.

1.11.6 IIR-Filter

IIR-Filter bilden im Zeitbereich rekursive Differenzengleichungen mit konstanten Koeffizienten. Als Beispiel soll an dieser Stelle die rekursive Variante unserer schon bekannten Mittelwertbildung aus fünf Meßwerten dienen.

$$y(k) = y(k-1) + \frac{x(k)}{5} - \frac{x(k-5)}{5}$$

$$Y(z) = z^{-1} \cdot Y(z) + \frac{1}{5} \cdot X(z) - \frac{1}{5} \cdot z^{-5} \cdot X(z)$$

$$\frac{Y(z)}{X(z)} = z^{-1} \cdot \frac{Y(z)}{X(z)} + \frac{1}{5} - \frac{1}{5} \cdot z^{-5}$$

$$H(z) = z^{-1} \cdot H(z) + \frac{1}{5} - \frac{1}{5} \cdot z^{-5}$$

$$H(z) \cdot (1 - z^{-1}) = \frac{1}{5} - \frac{1}{5} \cdot z^{-5}$$

$$H(z) = \frac{1/5 - 1/5 \cdot z^{-5}}{1 - z^{-1}}$$

Natürlich kann auch hier allgemeiner formuliert werden und man erhält die rekursive Differenzengleichung und die daraus resultierende Übertragungsfunktion im z -Bereich.

$$y(k) = a_0 \cdot x(k) + a_1 \cdot x(k-1) + a_2 \cdot x(k-2) + \dots + a_M \cdot x(k-M) \\ + b_1 \cdot y(k-1) + b_2 \cdot y(k-2) + \dots + b_N \cdot y(k-N)$$

$$H(z) = \frac{\sum_{k=0}^M a_k \cdot z^{-k}}{1 - \sum_{k=1}^N b_k \cdot z^{-k}} = \frac{a_0 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2} + \dots + a_M \cdot z^{-M}}{1 - b_1 \cdot z^{-1} + b_2 \cdot z^{-2} + \dots + b_N \cdot z^{-N}} \quad (1.47)$$

Die Übertragungsfunktion $H(z)$ bildet eine gebrochen rationale Funktion in Abhängigkeit von den konstanten Koeffizienten a_k und b_k . Sie bestimmen das Filterverhalten im Zeit- und Frequenzbereich. Hierbei sind die Koeffizienten b_k die Gewichtungen der vorhandenen Rückkopplungswege. Genau diese Rückkopplungen sind es aber, die rekursive Filter potentiell instabil machen. Es entspricht der gängigen Vorstellung, daß ein *stabiles*

System, wenn es kurzzeitig angeregt wird, nach einer gewissen Zeit immer wieder in eine stabile Ruhelage zurückkehrt. Sind jedoch Rückkopplungen vorhanden, kann es im ungünstigsten Fall vorkommen, daß die auf den Eingang rückwirkende Systemantwort zu einem unerwünschten Aufschaukeln des Systems führt. Jeder von uns kennt wohl das Phänomen der akustischen Rückkopplung bei einer Verstärkeranlage. Kommt man hier mit dem Mikrofon zu nahe an den Lautsprecher, dann macht sich die resultierende *Instabilität* als unangenehmes Pfeifen bemerkbar. Instabilitäten entstehen immer dann, wenn das Rückkopplungssignal das externe Eingangssignal positiv verstärkt.

Für Untersuchungen bezüglich der Stabilität eines Filters im Frequenzbereich ist es notwendig, das Nennerpolynom der Übertragungsfunktion näher zu betrachten. Wird der Nenner Null, dann geht $H(z)$ gegen Unendlich. Die Nullstellen des Nenners, *Polstellen* genannt, kennzeichnen also den Stabilitätsbereich des Filters und sind daher von besonderem Interesse. Allerdings sind sie in der Darstellung von Gl. 1.47 nur sehr schwer zu erkennen, weshalb wir eine geeignete Umformung wählen. Zunächst definieren wir $g = \max\{M, N\}$ und erweitern anschließend Zähler und Nenner mit z^g .

$$H(z) = \frac{a_0 \cdot z^g + a_1 \cdot z^{g-1} + a_2 \cdot z^{g-2} + \dots + a_M \cdot z^{g-M}}{z^g - b_1 \cdot z^{g-1} + b_2 \cdot z^{g-2} + \dots + b_N \cdot z^{g-N}}$$

Mit diesem Schritt ist eigentlich noch nichts gewonnen. Die Exponenten sind jetzt alle positiv und man erkennt, daß Zähler und Nenner gewöhnliche Polynome sind, mit einer dem Polynomgrad entsprechenden Anzahl von Nullstellen. Bestimmt man diese Nullstellen, läßt sich $H(z)$ auch in Produktform darstellen.

$$H(z) = \frac{a_0 \cdot (z - z_{01}) \cdot (z - z_{02}) \dots (z - z_{0M})}{(z - z_{x1}) \cdot (z - z_{x2}) \dots (z - z_{xN})} \quad (1.48)$$

In dieser Darstellung sind z_{0k} die Nullstellen und z_{xk} die Polstellen des Filters. Aufgrund der reellen Filterkoeffizienten sind sie entweder auch reell oder liegen paarweise konjugiert komplex vor. Dabei kann es durchaus vorkommen, daß Pol- und Nullstellen gleich sind und sich gegenseitig herauskürzen. Sie sind zudem durch die Filterkoeffizienten eindeutig festgelegt, weshalb die Vermutung nahe liegt, daß der Frequenzgang auch direkt aus ihnen bestimmt werden kann. Tatsächlich ist dem so.

Entwurf und Analyse von IIR-Filtern

Man kann sich leicht vorstellen, daß es mehrere Vorgehensweisen gibt, IIR-Filter mit definierten Eigenschaften zu entwerfen. Der klassische Weg besteht bekanntlich darin, daß man sich das gewünschte Übertragungsverhalten vorgibt und hieraus ein Filter entwirft, das dieser Charakteristik möglichst nahe kommt. Speziell für rekursive Filter seien an dieser Stelle die Impulsinvarianzmethode und die Bilineartransformation erwähnt. Beide Verfahren lassen sich aber nicht annähernd so leicht implementieren, wie die nachfolgend vorgestellten Methoden.

Der Pol-Nullstellen-Plan

Eine besonders anschauliche geometrische Interpretation des komplexen Frequenzganges nach Gln. (1.48) erhält man, wenn man die Lage der Pole und Nullstellen in der z -Ebene betrachtet und dabei die Frequenz entlang des Einheitskreises laufen läßt. Für ein Filter zweiter Ordnung mit zwei Pol- und Nullstellen ist dies hier einmal aufgezeigt.

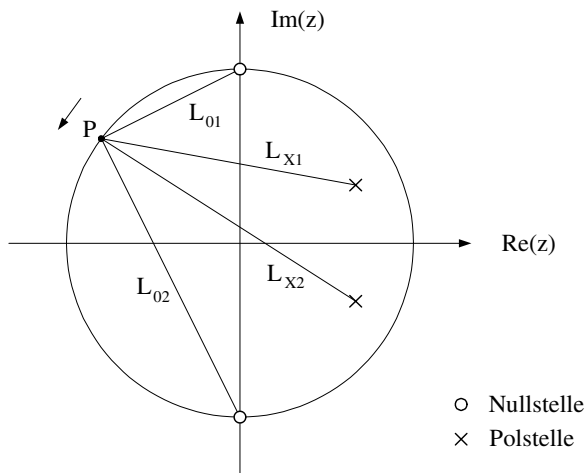


Abb. 1.44. Interpretation des Frequenzganges aus dem Pol-Nullstellen-Plan

Die einzelnen Faktoren der Zähler- und Nennerprodukte in der Produktform von $H(z)$ lassen sich als Differenzzeiger auffassen, die jeweils von den ortsfesten Polstellen bzw. Nullstellen zu dem veränderlichen Punkt P führen. Mit zunehmender Frequenz läuft dieser Punkt entgegen dem Uhrzeigersinn auf dem Einheitskreis entlang, wobei die Zeiger ihre Länge und Richtung ähnlich einem Gummiband verändern. Den Amplitudengang des Filters gewinnt man einfach durch Betragsbildung und Multiplikation der

einzelnen Zeiger. Aus der Summation der Zeigerwinkel, von der reellen Achse aus gesehen im mathematisch positiven Sinn, kann anschließend noch der Phasengang bestimmt werden. Amplitudengang und Phasengang bilden zusammen schließlich den komplexen Frequenzgang des Filters.

Um das hier beschriebene Verfahren als Algorithmus zu realisieren, ist nur noch ein wenig elementare Vektorrechnung notwendig. Der Verlauf des frequenzabhängigen Punktes P ergibt sich mit Hilfe der Parameterdarstellung des Kreises. An dieser Stelle kommt die Abtastfrequenz f_a ins Spiel. Sie dient lediglich zur Normierung und sorgt dafür, daß unsere Frequenz den Halbkreis von $0.. \pi$ durchläuft.

$$\vec{P}(f) = \begin{pmatrix} \cos(2\pi f / f_a) \\ \sin(2\pi f / f_a) \end{pmatrix} \text{ mit } 0 \leq f \leq \frac{f_a}{2}$$

$$L_i = \left| \vec{PZ}_i \right| = \sqrt{(\Re(\vec{P}) - \Re(\vec{Z}_i))^2 + (\Im(\vec{P}) - \Im(\vec{Z}_i))^2}$$

Ausgangspunkt der Implementierung ist aber zunächst die Schaffung einer geeigneten Struktur für die Datenhaltung, einmal für die Darstellung komplexer Zahlen und hierauf aufbauend eine Klasse für die Pol- und Nullstellen (auch Wurzeln genannt).

```
struct komplex// für komplexe Zahlen
{
    public double re;
    public double im;
}
```

```
class wurzel      // für Pol- und Nullstellen
{
    public int anzahl;
    public komplex[] z;

    public wurzel(int dim)
    {
        this.anzahl = 0;
        this.z = new komplex[dim];
    }
}
```

```
class Filter
{
    ...
    const int MaxPole = 10; // max. Zahl Polstellen
    const int MaxZero = 10; // max. Zahl Nullstellen
```

```

public wurzel Null = new wurzel(MaxZero);
public wurzel Pol  = new wurzel(MaxPole);
...

public void PolNullFrequenzgang()
{
    double[] Lo    = new double[Null.anzahl];
    double[] Lx    = new double[Pol.anzahl];
    double Lre     = 0; // Zeiger-Realteil
    double Lim     = 0; // Zeiger-Imaginärteil
    double Po      = 0; // Produkt der Nullstellenzeiger
    double Px      = 0; // Produkt der Polstellenzeiger
    double fa      = 1.0; // normierte Abtastfrequenz
    double startf  = 0.0; // Startfrequenz
    double stopf   = 0.5; // Stopfrequenz
    double deltaf  = (stopf-startf)/(Abtastwerte-1);
    double freq    = startf;

    for (int i=0; i<Abtastwerte; i++)
    {
        Po = 1;
        Px = 1;
        PS[i] = 0;

        for (int k=0; k<Null.anzahl; k++)
        {
            // Differenzzeiger aller Nullstellen berechnen
            Lre = Math.Cos(2*PI*freq/fa)-Null.z[k].re;
            Lim = Math.Sin(2*PI*freq/fa)-Null.z[k].im;
            // Betragsbildung
            Lo[k] = Math.Sqrt(Lre*Lre + Lim*Lim);
            // Multiplikation der Zeigerbeträge
            Po *= Lo[k];
            // Summation der Zeigerwinkel für den Phasengang
            PS[i] += Math.Atan2(Lim, Lre);
        }

        for (int k=0; k<Pol.anzahl; k++)
        {
            // Differenzzeiger aller Polstellen berechnen
            Lre = Math.Cos(2*PI*freq/fa)-Pol.z[k].re;
            Lim = Math.Sin(2*PI*freq/fa)-Pol.z[k].im;
            // Betragsbildung
            Lx[k] = Math.Sqrt(Lre*Lre + Lim*Lim);
            // Multiplikation der Zeigerbeträge
            Px *= Lx[k];
            // Summation der Zeigerwinkel für den Phasengang

```

```

    PS[i] -= Math.Atan2(Lim, Lre);
}

AS[i] = Po/Px;

freq += deltax;
}
}
...
}

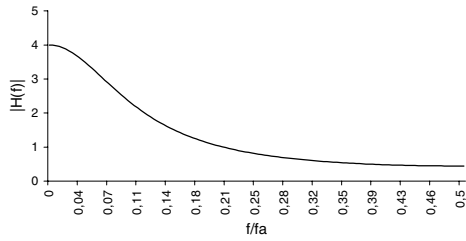
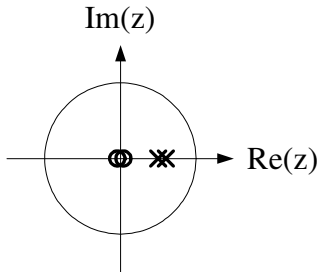
```

Auf der Basis manuell eingegebener Pol- und Nullstellen kann somit der resultierende Frequenzgang berechnet werden, wobei leichte Variationen in der Eingabe sich entsprechend im Frequenzgang niederschlagen. Die gezielte und iterative Umsetzung dieses Sachverhaltes erlaubt es einem Anwender, ein wenig Erfahrung ist hier natürlich sehr hilfreich, ein Filter mit den gewünschten Eigenschaften zu entwerfen. Hält man sich noch zusätzlich an ein paar einfache Grundregeln, erleichtert das die Arbeit ganz erheblich.

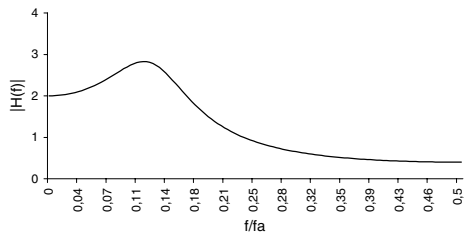
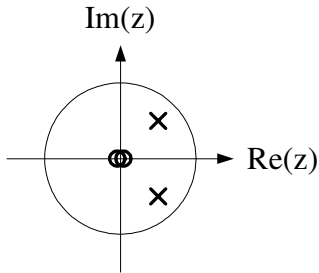
- Für den Entwurf stabiler Filter ist es notwendig, daß ausnahmslos alle Polstellen im inneren des Einheitskreises liegen. Ein außerhalb des Einheitskreises liegender Pol hat eine exponentiell ansteigende Impulsantwort (und damit Instabilität) zur Folge.
- Polstellen und Nullstellen sind immer entweder reell oder treten konjugiert komplex auf. Diese Einschränkung resultiert aus den reellen Koeffizienten, die für ein realisierbares Filter zwingend sind.
- Je näher die Nullstellen am Einheitskreis liegen, desto kleiner ist deren Abstand zum frequenzabhängigen Punkt P und desto mehr tragen sie zur Bedämpfung der jeweiligen Frequenz bei.
- Polstellen in der Nähe des Einheitskreises führen dagegen zu einer Anhebung des Amplitudenganges, da das Nennerprodukt bei der betreffenden Frequenz dann sehr klein wird.

Einer Überprüfung dieser Regeln steht nun nichts mehr im Wege. Ausgangspunkt für die weiteren Untersuchungen sollen Filter zweiter Ordnung sein, wie sie als Grundbausteine beim Filterentwurf typisch sind. In den folgenden Beispielen sollte man sich nicht mehr über die doppelte Nullstelle im Koordinatenursprung wundern. Sie resultiert aus der Erweiterung von Zähler und Nenner von $H(z)$ mit z^2 . Ist nur der Amplitudenfrequenzgang von Interesse, könnte man sie eigentlich auch weggelassen. Tatsächlich haben Pol- und Nullstellen im Koordinatenursprung keinen Einfluß auf den Amplitudenfrequenzgang, sondern wirken sich nur auf den Phasenfrequenzgang aus, wie man sich leicht mit Hilfe von Abb. 1.44 überlegen kann.

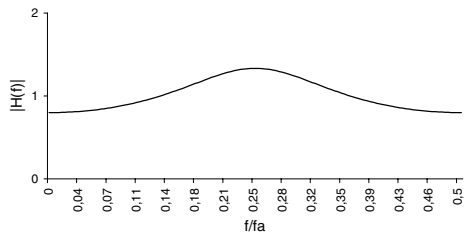
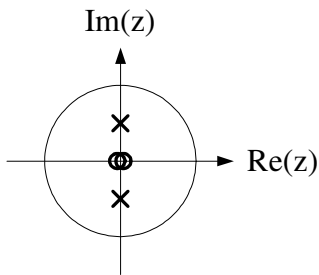
Tabelle 1.7. Einfluß der Polstellen auf das Übertragungsverhalten



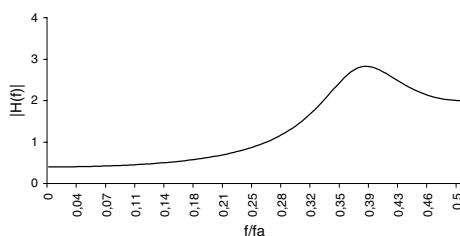
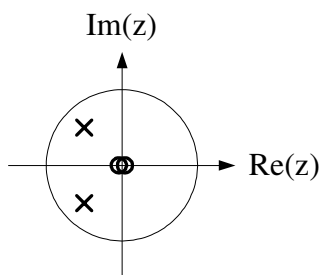
$$H(z) = \frac{1}{1 - z^{-1} + 0.25 \cdot z^{-2}}$$



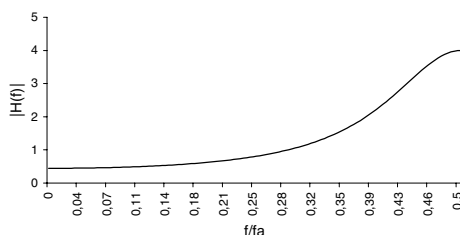
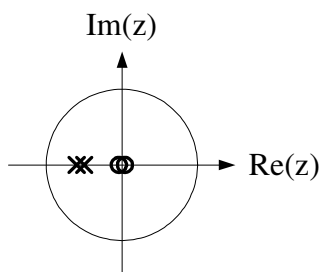
$$H(z) = \frac{1}{1 - z^{-1} + 0.5 \cdot z^{-2}}$$



$$H(z) = \frac{1}{1 + 0.25 \cdot z^{-2}}$$

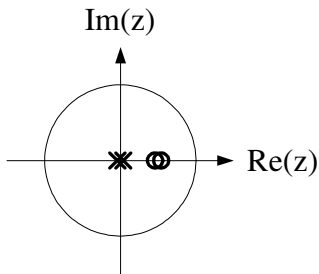


$$H(z) = \frac{1}{1 + z^{-1} + 0.5 \cdot z^{-2}}$$

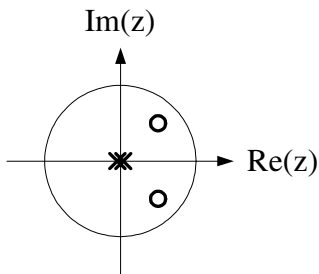
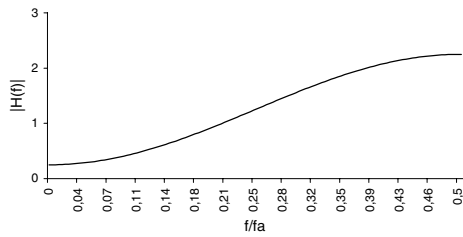


$$H(z) = \frac{1}{1 + z^{-1} + 0.25 \cdot z^{-2}}$$

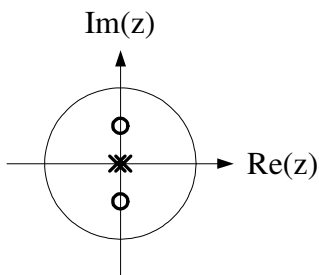
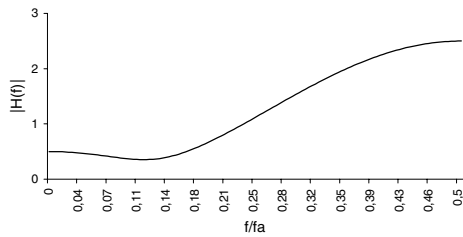
Selbstverständlich können wir FIR-Filter als Sonderfall der allgemeinen IIR-Filter ebenfalls mit dieser Methode entwerfen. Wir erinnern uns, daß FIR-Filter zwar auch Polstellen besitzen, diese aber alle im Koordinatenursprung der z -Ebene liegen. Nachfolgend dargestellt ist daher ihr Übertragungsverhalten bei Variation der Nullstellen. Die unten aufgeführten Beispiele entstehen durch Vertauschen von Polstellen und Nullstellen in Tabelle 1.7. Im Ergebnis erhält man den jeweils komplementären Filtertyp. Aus einem Tiefpaß wird somit ein Hochpaß, aus einem Bandpaß wird eine Bandsperrung und umgekehrt.

Tabelle 1.8. Einfluß der Nullstellen auf das Übertragungsverhalten

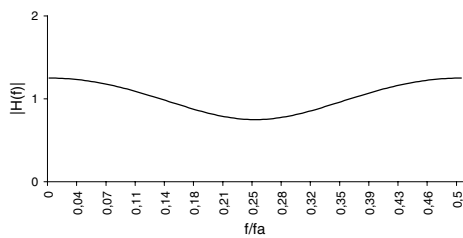
$$H(z) = 1 - z^{-1} + 0.25 \cdot z^{-2}$$

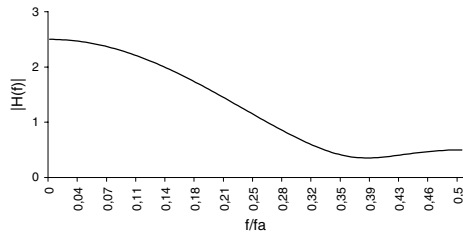
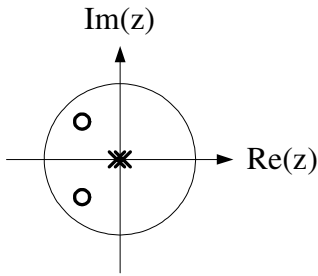


$$H(z) = 1 - z^{-1} + 0.5 \cdot z^{-2}$$

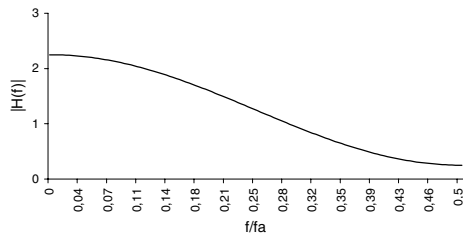
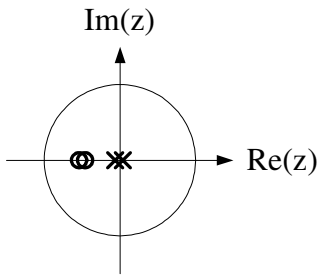


$$H(z) = 1 + 0.25 \cdot z^{-2}$$





$$H(z) = 1 + z^{-1} + 0.5 \cdot z^{-2}$$



$$H(z) = 1 + z^{-1} + 0.25 \cdot z^{-2}$$

Von den IIR-Koeffizienten zum Frequenzgang

Mit dem vorigen Abschnitt beschriebenen Verfahren verfügt man über ein für den Filterentwurf und die Filteranalyse gleichermaßen geeignetes Werkzeug. Trotzdem ist der Gedanke, daß dies nur über die Polstellen und Nullstellen der komplexen Übertragungsfunktion möglich sein soll, doch irgendwie unbefriedigend. Vielmehr wird man völlig zu recht erwarten, den Frequenzgang von IIR-Filtern auch direkt aus ihren Filterkoeffizienten berechnen zu können. Das hat schließlich auch schon bei den FIR-Filtern funktioniert.

IIR-Filter bestehen aus zwei Signalzweigen, die sich völlig unabhängig voneinander betrachten lassen. Zum einen der Vorwärtszweig (Zähler von $H(z)$), der den FIR-Anteil repräsentiert, zum anderen der Rückkopplungszweig (Nenner von $H(z)$). Was für den Vorwärtszweig schon möglich war, sollte doch auch für den Rückkopplungsteil gehen. Natürlich sind beide Zweige nicht ganz gleichberechtigt, da es schließlich reine Vorwärtsfilter gibt, wogegen jedoch ein reines Rückkopplungsfilter aufgrund der dann fehlenden Eingangswerte keinen Sinn macht. Trotz dieser Einschränkung

kann die Übertragungsfunktion von IIR-Filtern als Quotient zweier eigenständiger Filterfunktionen $H_Z(z)$ und $H_N(z)$ gesehen werden.

$$H(z) = \frac{H_Z(z)}{H_N(z)} = \frac{\sum_{k=0}^M a_k \cdot z^{-k}}{\sum_{k=0}^M b_k \cdot z^{-k}}$$

Wird auf b_0 normiert und läßt man auch noch unterschiedliche Filterordnungen für Zähler und Nenner zu, dann entspricht dieser Ausdruck der allgemeinen Übertragungsfunktion, wie wir sie schon in Gln. 1.47 definiert haben. Die Abweichung an dieser Stelle bedeutet aber keine Beschränkung der Allgemeinheit. Selbstverständlich läßt sich jetzt separat für beide Zweige durch z -Transformation der Impulsantwort (sie ist identisch mit den jeweiligen Koeffizienten a_k , b_k) das gesuchte Übertragungsverhalten berechnen. Das sich schließlich der Frequenzgang $H(f)$ des Gesamtsystems durch Quotientenbildung der Teilfrequenzgänge ergibt, entspricht wohl der allgemeinen Erwartung. Genau wie schon bei den FIR-Filtern, werden wir jedoch anstelle der z -Transformation die viel leichter zu implementierende DFT verwenden.

$$|H(f)| = \frac{|H_Z(f)|}{|H_N(f)|} = \frac{DFT(a_k)}{DFT(b_k)} \quad (1.49)$$

$$\varphi(f) = \varphi_Z(f) - \varphi_N(f) \quad (1.50)$$

```
class Filter
{
    ...
    // Filterkoeffizienten des Vorwärtzweiges
    public double[] a = new double[Abtastwerte];
    // Filterkoeffizienten des Rückkopplungszweiges
    public double[] b = new double[Abtastwerte];
    // Amplitudengang des Zählers von H(z)
    public double[] Az = new double[Abtastwerte];
    // Amplitudengang des Nenners von H(z)
    public double[] An = new double[Abtastwerte];
    // Phasengang des Zählers von H(z)
    public double[] Pz = new double[Abtastwerte];
    // Phasengang des Nenners von H(z)
    public double[] Pn = new double[Abtastwerte];
    // Amplitudengang von H(z)
    public double[] AS = new double[Abtastwerte];
```

```
// Phasengang von H(z)
public double[] PS = new double[Abtastwerte];
...

public void ErzeugeTestsignal_2()
{
    // Filterkoeffizienten des Vorwärtzweiges
    a[0] = 1;
    a[1] = 0;
    a[2] = 0;

    // Filterkoeffizienten des Rückkopplungszweiges
    b[0] = 1;
    b[1] = -1;
    b[2] = 0.5;

    for (int k=3; k<Abtastwerte; k++)
    {
        a[k] = 0;
        b[k] = 0;
    }
}
...

public void FilterFrequenzgang(double StartFreq,
                               double StopFreq)
{
    double deltaf = (StopFreq-StartFreq)/
                    (Abtastwerte-1);
    double freq    = StartFreq;
    double real    = 0;
    double imag    = 0;

    // über alle Frequenzen
    for (int i=0; i<Abtastwerte; i++)
    {
        real = 0;
        imag = 0;

        // über alle Vorwärtskoeffizienten a[k]
        for (int k=0; k<Abtastwerte; k++)
        {
            real += a[k]*Math.Cos(2*PI*freq*k/Abtastwerte);
            imag += a[k]*Math.Sin(2*PI*freq*k/Abtastwerte);
        }
        // Amplituden- und Phasengang -> Zähler von H(z)
        Az[i] = Math.Sqrt(real*real + imag*imag);
        Pz[i] = Math.Atan2(imag, real);
    }
}
```

```

real = 0;
imag = 0;

// über alle Rückkopplungskoeffizienten b[k]
for (int k=0; k<Abtastwerte; k++)
{
    real += b[k]*Math.Cos(2*PI*freq*k/Abtastwerte);
    imag += b[k]*Math.Sin(2*PI*freq*k/Abtastwerte);
}
// Amplituden- und Phasengang -> Nenner von H(z)
An[i] = Math.Sqrt(real*real + imag*imag);
Pn[i] = Math.Atan2(imag, real);

// Gesamtsystem -> H(z) = Hz(z)/Hn(z)
if (An[i] > 0)
{
    // IIR-Filter
    AS[i] = Az[i]/An[i];
}
else
{
    // FIR-Filter
    AS[i] = Az[i];
}
PS[i] = Pz[i]-Pn[i];

freq += deltaf;
}
}
...
}

```

1.11.7 Der Phasengang

In unseren Betrachtungen digitaler Systeme stand bisher ausschließlich der Amplitudengang im Vordergrund. Aber erst Amplitudengang und Phasengang gemeinsam beschreiben die Frequenzabhängigkeit eines Systems vollständig, bilden also den Frequenzgang $H(f)$ entsprechend Gl. (1.36).

Üblicherweise verändern Signale beim Durchlaufen digitaler Systeme nicht nur ihre spektrale Zusammensetzung, sie werden zusätzlich auch mehr oder weniger stark verzögert, d. h. das Signal am Systemausgang eilt dem am Eingang anliegenden Signal nach. Die Verschiebung des (sinusförmigen) Ausgangssignals gegenüber dem (sinusförmigen) Eingangssignal wird gerade durch den *Phasengang* beschrieben.

Neben der gezielten spektralen Veränderungen eines Signals durch frequenzselektive Filter, also die Verstärkung oder Unterdrückung bestimmter Frequenzanteile, wirkt auch dessen Phasengang auf den Signalverlauf. Eine solche zusätzliche Abhängigkeit, die sich als laufzeitbedingte Verzerrung bemerkbar macht, ist in den meisten Anwendungen aber unerwünscht. Man greift daher bevorzugt auf solche Filter zurück, deren Phasengang keinen Einfluß auf die Signalform hat. Allgemein bezeichnet man Filter mit dieser Eigenschaft als *linearphasig*. FIR-Filter, mit beidseitig um ein Zentralelement symmetrischen Filterkoeffizienten, besitzen gerade die Eigenschaft der Linearphasigkeit. IIR-Filter dagegen haben meist einen nichtlinearen Phasengang, was bei der Übertragung von Pulsen zu einer asymmetrischen Verschiebung des ursprünglich symmetrischen Signals führt.

1.11.8 Vergleich zwischen FIR- und IIR-Filtern

Nachdem wir uns in den vorangegangenen Abschnitten ausführlich mit FIR- und IIR-Filtern beschäftigt haben, ist deutlich geworden, daß sich beide Filterarten in vielerlei Hinsicht unterscheiden.

FIR-Filter besitzen keinen Rückkopplungszweig und sind aus diesem Grund auch für beliebige Koeffizienten immer stabil. Da ihre Impulsantwort grundsätzlich frei wählbar ist, haben sie unter Umständen einen Frequenzgang, für den kein entsprechendes analoges Filter existiert. Möchte man einen vorgegebenen Frequenzgang mit einem FIR-Filter nachbilden, ist das oft nur mit sehr langen Filtern hoher Ordnung möglich. Als Softwareimplementierung beansprucht es dann auch viel Speicherplatz und verursacht relativ lange Rechenzeiten. Ein wesentlicher Vorteil von FIR-Filtern (neben ihrer Stabilität) ist der Umstand, daß es mit ihnen leicht möglich ist einen linearen Phasengang und eine konstante Gruppenlaufzeit zu realisieren, was sie unter anderem für Anwendungen im Audio-Bereich sehr attraktiv macht.

IIR-Filter haben in den meisten Fällen ein analoges Gegenstück, lassen sich also auch mit Hilfe von Operationsverstärkern, Widerständen und Kondensatoren als elektronische Schaltung realisieren. Sie kommen daher vorzugsweise dann zum Einsatz, wenn es darum geht die Eigenschaften eines bekannten Analogfilters durch ein Digitalfilter nachzubilden. Die notwendige Filterordnung ist zudem, bei vergleichbarer Selektivität, deutlich geringer als bei einem FIR-Filter, was den Speicherbedarf und die erforderliche Rechenzeit stark reduziert. Ein latentes Problem bei IIR-Filtern ist deren Stabilität, insbesondere dann, wenn ihre Polstellen in der z -Ebene nahe am Einheitskreis liegen. Selbst wenn das entworfene Filter im Einsatz stabil ist, kann es doch vorkommen, daß das tatsächliche Übertragungsverhalten nach

der Implementierung durch Rundungs- oder Abschneidefehler der Koeffizienten vom ursprünglichen Entwurf abweicht.

1.11.9 FFT-Filter

Egal ob wir bisher mit FIR-Filtern oder mit IIR-Filtern gearbeitet haben, eines war immer klar, die Filterung ist eine mathematische Operation, die auf die Eingangswerte $x(k)$ im Zeitbereich angewendet wird. Betrachtungen im Frequenzbereich waren dagegen nur ein zusätzliches Hilfsmittel, um die erforderlichen Eigenschaften zu definieren oder unbekannte Eigenschaften zu analysieren. Mit dem jetzt vorgestellten *FFT-Filter* kehren sich die Verhältnisse um. In der Regel definieren sich Filter über ihr Verhalten im Frequenzbereich. Konsequenterweise gibt es deshalb auch Entwurfsverfahren, z. B. die Fenstermethode, bei denen man einen gewünschten Frequenzgang einfach vorgeben kann. Leider kränkelte die Umsetzung dieses an sich doch richtigen Ansatzes bisher an der Notwendigkeit die Eigenschaften aus dem Frequenzbereich in den Zeitbereich übertragen zu müssen. Und damit geht der Ärger los. Das gewünschte Frequenzverhalten in reale Filterkoeffizienten abzubilden ist nämlich gar nicht so einfach, wie wir gesehen haben. Man fängt sich eine Reihe von nichtidealen Effekten ein, die sich nur mit sehr viel Aufwand wieder unterdrücken lassen. Da wäre es doch viel günstiger die Filterung ebenfalls im Frequenzbereich ausführen zu können, um diese Schwierigkeiten zu umgehen. Genau das ist mit FFT-Filtern möglich.

Zunächst werden die Eingangswerte mittels FFT in den Frequenzbereich transformiert. Anschließend wird das resultierende Spektrum mit dem gewünschten Frequenzgang multipliziert. Dabei kann der Anwender grundsätzlich, ähnlich wie bei einem Equalizer, beliebige Verläufe wählen. In einem letzten Schritt ist das Ergebnis dann wieder mit Hilfe der IFFT in den Zeitbereich zu transformieren. Mit diesem Vorgehen lassen sich Übergangsbereich und Sperrdämpfungen realisieren, wie das mit gewöhnlichen FIR- oder IIR-Filtern kaum möglich wäre. Aus diesem Grund sind FFT-Filter besonders in der Audio-Signalverarbeitung weit verbreitet.

Tabelle 1.9. Vergleich zwischen FIR-Filtern und IIR-Filtern

Merkmal	FIR-Filter	IIR-Filter
Selektivität	gering	hoch
Erforderliche Ordnung	hoch	gering
Speicherbedarf	hoch	gering
Rechenzeitbedarf	hoch	gering
Lineare Phase	leicht möglich	kaum möglich
Stabilität	immer	bedingt
Genauigkeit der Koeffizienten	mäßig	hoch

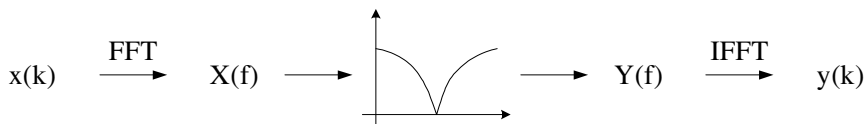


Abb. 1.45. Funktionsprinzip eines FFT-Filters

1.12 Experimentelle Systemanalyse

Schon bei der Einführung der Übertragungsfunktion und des Frequenzganges im Abschn. 1.11.3 wurde darauf hingewiesen, daß es in der Meß- und Regelungstechnik üblich ist, ein unbekanntes Übertragungssystem mit geeigneten Testfunktionen zu beaufschlagen, um aus der Systemantwort auf dessen dynamische Eigenschaften zu schließen. Mit unseren jetzt erweiterten Kenntnissen können wir endlich die Möglichkeiten der *experimentellen Systemanalyse* nutzen. Die Bestimmung der Systemparameter wird auch *Identifikation* genannt. Als typische Übertragungssysteme in der Signalverarbeitung sind digitale Filter geeignete Testobjekte.

Wir sind also nun in der Lage, digitale Filter auf der Basis ihrer Koeffizienten oder ihrer Pol- und Nullstellen zu berechnen. Damit sind sowohl der Zeit-, als auch der Frequenzbereich abgedeckt. Allerdings ist es in beiden Fällen zwingend notwendig, die Koeffizienten oder die Pol- und Nullstellen auch tatsächlich zu kennen, also Zugriff auf die „Innereien“ des Filters zu haben. Es sind aber durchaus Anwendungen denkbar, bei denen das nicht sichergestellt ist, zum Beispiel in verteilten Systemen oder bei der Verwendung von eingekauften Fremdkomponenten. Das Filter ist dann beispielsweise nur über Schnittstellen zugänglich, bildet also eine Black-Box. Sich in solchen Fällen blind auf zugesagte Eigenschaften zu verlassen könnte jedoch sehr teuer werden.

1.12.1 Identifikation im Frequenzbereich

In der Praxis kann es also durchaus zweckmäßig sein, die Eigenschaften eines Systems (oder von Teilen des Systems) von außen zu bestimmen. Aber welche systembeschreibenden Größen stehen hierfür zur Verfügung, wenn doch der direkte Zugriff auf das Innenleben nicht möglich ist? Ein Rückblick auf die Abb. 1.26 und 1.27 sollte die Antwort geben. Natürlich steckt die notwendige Information in dem zugeführten Eingangssignal $x(k)$ und dem resultierenden Ausgangssignal $y(k)$, der Systemantwort. Beide

zusammen beschreiben das Übertragungsverhalten eines unbekannten (linearen) Systems vollständig.

$$y(k) = h(k) * x(k)$$

$$Y(z) = H(z) \cdot X(z)$$

$$H(z) = \frac{DFT(y(k))}{DFT(x(k))}$$

```
class Identifikation
{
    const int Abtastwerte = 512;
    const double PI      = 3.141592653589793;
    ...
    // Eingangssignal im Zeitbereich
    public double[] x = new double[Abtastwerte];
    // Ausgangssignal im Zeitbereich
    public double[] y = new double[Abtastwerte];
    // Eingangssignal im Frequenzbereich
    public double[] XA = new double[Abtastwerte];
    public double[] XP = new double[Abtastwerte];
    // Ausgangssignal im Frequenzbereich
    public double[] YA = new double[Abtastwerte];
    public double[] YP = new double[Abtastwerte];
    // Amplitudengang und Phasengang des Testsystems
    public double[] HA = new double[Abtastwerte];
    public double[] HP = new double[Abtastwerte];
    ...
    public void Frequenzgang(double StartFreq,
                             double StopFreq)
    {
        double deltaf = (StopFreq-StartFreq)/
                        (Abtastwerte-1);
        double freq   = StartFreq;
        double real    = 0;
        double imag    = 0;

        // über alle Frequenzen
        for (int i=0; i<Abtastwerte; i++)
        {
            real = 0;
            imag = 0;

            // DFT des Eingangssignals
            for (int k=0; k<Abtastwerte; k++)
            {
```

```

        real += x[k]*Math.Cos(2*PI*freq*k/Abtastwerte);
        imag += x[k]*Math.Sin(2*PI*freq*k/Abtastwerte);
    }
    real /= Abtastwerte;
    imag /= Abtastwerte;
    XA[i] = Math.Sqrt(real*real + imag*imag);
    XP[i] = Math.Atan2(imag, real);

    real = 0;
    imag = 0;

    // DFT des Ausgangssignals
    for (int k=0; k<Abtastwerte; k++)
    {
        real += y[k]*Math.Cos(2*PI*freq*k/Abtastwerte);
        imag += y[k]*Math.Sin(2*PI*freq*k/Abtastwerte);
    }
    real /= Abtastwerte;
    imag /= Abtastwerte;
    YA[i] = Math.Sqrt(real*real + imag*imag);
    YP[i] = Math.Atan2(imag, real);

    // Amplitudengang des Übertragungssystems
    HA[i] = YA[i]/(XA[i] + 1.0E-12);
    // Phasengang des Übertragungssystems
    HP[i] = YP[i] - XP[i];

    freq += deltaf;
}
}
...
}

```

Als „unbekanntes“ Testsystem nehmen wir einfach ein Mittelwertfilter mit $M=10$ Abtastwerten. Da dessen Amplitudenfrequenzgang aus den bisherigen Betrachtungen schon bekannt sein sollte (Abb. 1.43), lassen sich die Ergebnisse unserer experimentellen Analyse leicht verifizieren.

```

class Identifikation
{
    ...
    public void Filterung()
    {
        int M = 10;
        double[] a = new double[M];
    }
}

```

```

// Mittelwertfilter als Übertragungssystem
for (int k=0; k<M; k++)
{
    a[k] = 1.0/(double)M;
}

// Filterung im Zeitbereich (= Faltung)
for (int k=M-1; k<Abtastwerte; k++)
{
    for (int j=0; j<M; j++)
    {
        y[k] += x[k-j] * a[j];
    }
}
...
}

```

Bis hierhin ist alles klar, man nimmt ein Eingangssignal $x(k)$, legt es an sein unbekanntes Testsystem an und zeichnet die Systemantwort $y(k)$ auf. Aus den Frequenztransformierten $X(f)$, $Y(f)$ beider Signale läßt sich dann der Frequenzgang $H(f)$ unseres Testsystems bestimmen, das dann nicht mehr ganz so unbekannt ist. Wie man sich aber denken kann, sind für das hier beschriebene Verfahren der Systemidentifikation nicht alle Eingangssignale gleichermaßen geeignet. Selbstverständlich ist ein Gleichsignal $x(k) = \text{const}$ am Eingang nicht in der Lage eine frequenzabhängige Systemantwort zu provozieren. Die Dynamik eines auf diese Art angeregten Systems bliebe also weiterhin verborgen. Aus diesem Grund greift man zur Messung des Übertragungsverhaltens auf bewährte und standardisierte *Testsignale* zurück.

Der Einheitsimpuls

Grundsätzlich sollte der *Einheitsimpuls* schon aus früheren Abschnitten bekannt sein. Regt man ein System mit dem Einheitsimpuls an, ergibt sich am Ausgang die Impulsantwort. Warum dieses eigentlich recht unscheinbare Testsignal so besonders aussagekräftig ist, wird sofort klar, wenn man sich erinnert, daß ein idealer Impuls Sinusschwingungen aller Frequenzen mit identischen Amplituden enthält. Das zu untersuchende System wird demzufolge mit allen Frequenzen gleichzeitig angeregt, allein durch eine einzige Messung. In der Analogtechnik ist der mittels Funktionsgenerator erzeugte Impuls als Testfunktion nicht ganz unproblematisch. Wegen seiner nur geringen Signalenergie bleibt die Impulsantwort oft unterhalb der

Nachweisgrenze. Erhöht man zum Ausgleich die Amplitude, läuft man Gefahr das Testsystem zu übersteuern. In der digitalen Signalverarbeitung spielen derartige Probleme dagegen keine Rolle. Wir können ein entsprechendes Testsignal leicht erzeugen und beliebig verschieben.

$$\delta(k) = \begin{cases} 1 & (k = 0) \\ 0 & (k \neq 0) \end{cases} \quad (1.51)$$

```
class Identifikation
{
    ...
    public void ErzeugeTestsignal_1()
    {
        for (int k=0; k<Abtastwerte; k++)
        {
            // Einheitsimpuls
            if (k == Abtastwerte/2)
            {
                x[k] = 1;
            }
            else
            {
                x[k] = 0;
            }
        }
    }
    ...
}
```

Unser so erzeugter Einheitsimpuls wird nun durch Aufruf der Methode `Filterung()` auf unser Testsystem geschaltet. Als Ergebnis erhalten wir die Systemantwort $y(k)$. Mit dem zugeführten Eingangssignal und dem daraus resultierenden Ausgangssignal verfügen wir im Zeitbereich über alle notwendigen systembeschreibenden Informationen. Jetzt sind lediglich noch beide Folgen einer DFT zu unterziehen, um das gesuchte Übertragungsverhalten im Frequenzbereich zu ermitteln.

$$x(k) = \delta(k)$$

$$y(k) = h(k) * \delta(k)$$

$$|H(f)| = \frac{DFT(h(k) * \delta(k))}{DFT(\delta(k))}$$

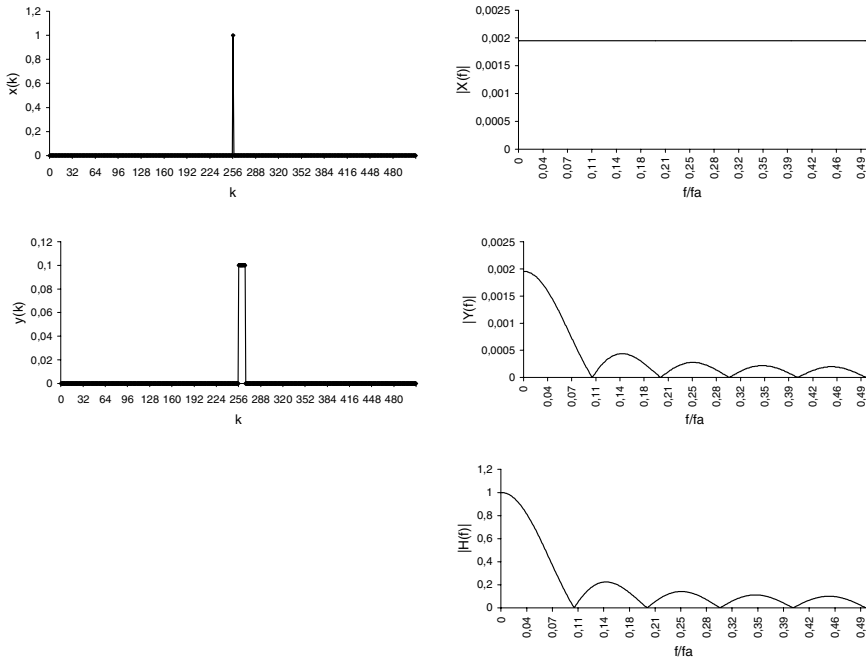


Abb. 1.46. Bestimmung der Übertragungsfunktion $|H(f)|$ mit Hilfe des Einheitsimpulses als Testsignal

Das erzielte Ergebnis kann eigentlich niemanden mehr überraschen. Ein Einheitsimpuls am Eingang unseres FIR-Filters ergibt als Filterantwort die Filterkoeffizienten selbst. Das durch sie gebildete Rechtecksignal hat als Spektrum eine Spaltfunktion, welche der Pulsweite entsprechend aufgefächert ist. Bei unserem unendlich schmalen Eingangspuls entartet diese Spaltfunktion zu $X(f) = \text{const.}$ Somit läßt sich im vorliegenden Fall der Verlauf von $H(f) = Y(f)/X(f)$ durch einige kurze Überlegungen verifizieren. In den nachfolgenden Beispielen ist das leider nicht mehr ganz so einfach.

Der Einheitssprung

Die in der Regelungstechnik am häufigsten benutzte Testfunktion ist der *Einheitssprung* $\sigma(t)$. Mathematisch gesehen handelt es sich hierbei um das Integral des Einheitsimpulses. Ein großer Vorteil der Sprungfunktion besteht darin, daß das Testsystem nur mit einer einzigen Zustandsänderung, von Null auf Eins, beaufschlagt wird (anders als beim Impuls, bei dem es zwei direkt aufeinanderfolgende Zustandsänderungen sind). Aus diesem

Grund läßt sich eine Sprungfunktion durch einen analogen Funktionsgenerator auch leichter erzeugen, als ein Impuls. Die resultierende Systemantwort wird als *Sprungantwort* oder *Übergangsfunktion* bezeichnet.

$$\sigma(k) = \begin{cases} 0 & (k \leq 0) \\ 1 & (k > 0) \end{cases} \quad (1.52)$$

Ganz frei von Problemen ist auch die digitale Signalverarbeitung nicht. Schauen wir doch einmal genauer auf die Definition der Sprungfunktion. Zunächst ist sie Null, bis sie zu einem bestimmten Zeitpunkt auf den Wert Eins springt und diesen beibehält. Auf den ersten Blick nicht weiter spektakulär, denkt man aber länger darüber nach, stellt sich die Frage, was kommt eigentlich nach dem Sprung? Oder noch viel wichtiger, wie interpretiert die DFT das Ende des Abtastintervalls? Die DFT geht implizit von periodischen Signalen aus, somit würde am Intervallende ein Rücksprung von Eins auf Null stehen. Unser Testsignal wäre dann ein Rechteck, dessen Länge von der ansteigenden Flanke (Zeitpunkt frei wählbar) bis zum Intervallende reicht. Unterschiedlich weite Rechtecke oder Pulse besitzen aber

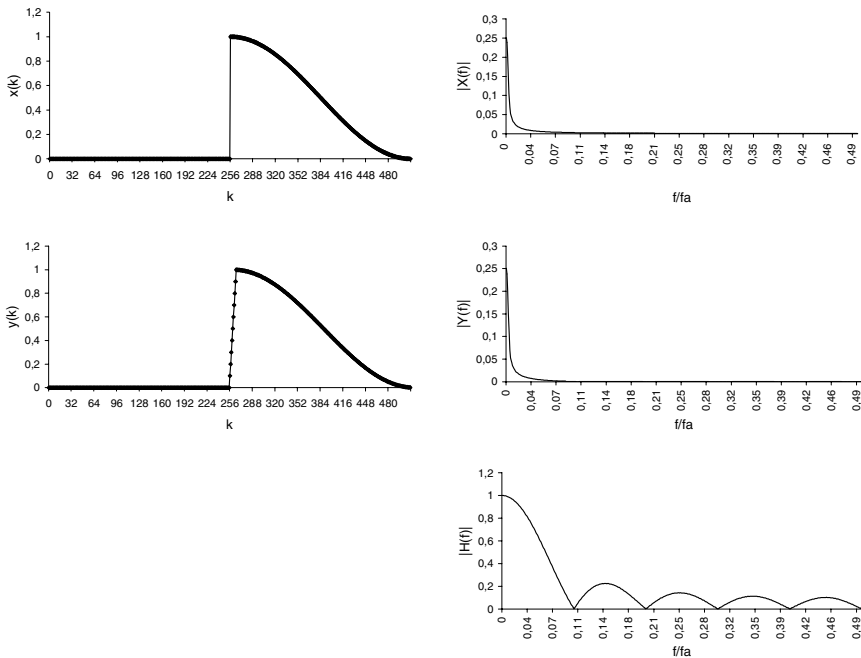


Abb. 1.47. Bestimmung der Übertragungsfunktion $|H(f)|$ mit Hilfe des Einheitssprungs als Testsignal

auch unterschiedliche Spektren, wie Abschn. 1.5 gezeigt hat. Für Untersuchungen im Frequenzbereich ist das jedoch ungünstig. Besser wäre es, den scheinbaren Rücksprung für die DFT irgendwie auszublenden. Abhilfe schafft auch hier wieder, wie sollte es auch anders sein, eine Fensterung. Wird die Dauer der Sprungfunktion durch ein zusätzliches Zeitfenster künstlich begrenzt, ist die Anwendung der DFT problemlos möglich. Wir schalten unserem Testsignal daher noch ein Hanning-Fenster nach. Für Untersuchungen im Zeitbereich sind solche Maßnahmen nicht notwendig. Dort reicht es aus, wenn die Eins nur lange genug ansteht, bis sich das Testsystem eingeschwungen hat. Was danach kommt interessiert nicht weiter.

```
class Identifikation
{
    ...
    public void ErzeugeTestsignal_2()
    {
        for (int k=0; k<Abtastwerte; k++)
        {
            // Sprungfunktion
            if (k <= Abtastwerte/2)
            {
                x[k] = 0;
            }
            else
            {
                x[k] = 1;
            }

            // zusätzliche Hanning-Fensterung
            x[k] *= (0.5 - 0.5*Math.Cos(2*PI*k/Abtastwerte));
        }
    }
    ...
}
```

$$x(k) = \sigma(k) \cdot w(k)$$

$$y(k) = h(k) * (\sigma(k) \cdot w(k))$$

$$|H(f)| = \frac{DFT(h(k) * (\sigma(k) \cdot w(k)))}{DFT(\sigma(k) \cdot w(k))}$$

Der Gleitsinus

Den Gleitsinus, auch Chirp genannt, haben wir schon in Abschn. 1.10 kennengelernt. Mit seiner kontinuierlich anwachsenden Frequenz erscheint es als Testsignal wie geschaffen für unsere Zwecke.

```
class Identifikation
{
    ...
    public void ErzeugeTestsignal_3()
    {
        double f = 0;

        for (int k=0; k<Abtastwerte; k++)
        {
            x[k] = Math.Sin(2*PI*f*k/Abtastwerte);

            f += 0.25;
        }
    }
    ...
}
```

$$x(k) = \text{chirp}(k)$$

$$y(k) = h(k) * \text{chirp}(k)$$

$$|H(f)| = \frac{DFT(h(k) * \text{chirp}(k))}{DFT(\text{chirp}(k))}$$

Das Spektrum des Gleitsinus kann durchaus auch anders aussehen, denn es hängt von der Schnelligkeit ab, mit der die Frequenz im Abtastintervall anwächst. Hiervon sollte man sich aber nicht beirren lassen. Wählt man den abgedeckten Frequenzbereich günstig, kann man den Frequenzgang des Systems sogar schon im Zeitbereich ablesen. Die gute Übereinstimmung von $y(k)$ mit $|H(f)|$ ist deutlich zu erkennen. Jedem Punkt im Zeitbereich kann die jeweilige Frequenz exakt zugeordnet werden. Durch diese besondere Anschaulichkeit unterscheidet sich der Gleitsinus von allen anderen Testsignalen.

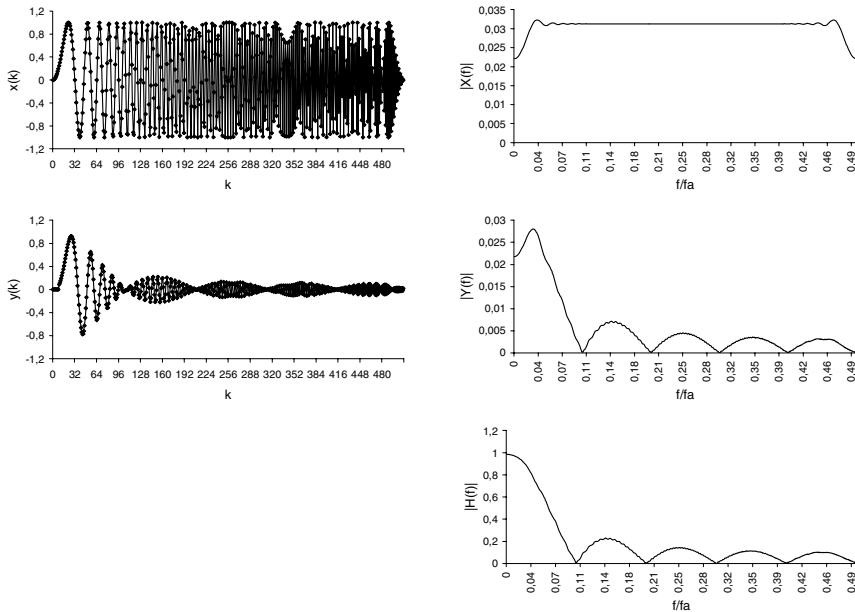


Abb. 1.48. Bestimmung der Übertragungsfunktion $|H(f)|$ mit Hilfe des Gleitsinus als Testsignal

Rauschen

Zugegeben, die Verwendung von stochastischem *Rauschen* als Testsignal klingt zunächst wie ein Widerspruch. Rauschen zeigt bekanntlich keine Erhaltungstendenz, d. h. es ist nicht möglich aus der Betrachtung eines beliebigen Intervalls auf vorherige oder nachfolgende Intervalle zu schließen. Der zeitliche Verlauf ist rein zufällig und unvorhersehbar. Sowenig es uns auch gefällt, Rauschen ist allgegenwärtig und macht sich in der Signalverarbeitung als dem Nutzsignal überlagerte Störung bemerkbar. Trotzdem hat es auch seine guten Seiten, zumindest als Testsignal.

Ideales *weißes Rauschen* als Signal trägt zwar selbst keinerlei Information, enthält dafür aber alle Frequenzen. Durchläuft es unser Testsystem (wird also mit einem Tiefpaß gefiltert), dann entsteht zunächst sogenanntes *farbiges Rauschen*. Seine fehlenden bzw. verbleibenden Frequenzanteile sind es, die schließlich die Dynamik des Testsystems preisgeben. An dieser Stelle ist mit der im Beispielcode verwendeten Klasse `Zufall` ein kleiner Vorgriff auf das nächste Kapitel notwendig, da natürlich die Standardsprachelemente von C# (und auch jeder anderen Programmiersprache) die Erzeugung von Rauschsignalen nicht unmittelbar unterstützen.

Problematisch ist hier lediglich die Auflösung. Bei normalverteiltem Rauschen ist die Streuung umgekehrt proportional zur Wurzel aus der Zahl der Abtastwerte. Für eine Verdopplung der Genauigkeit (bzw. halbierte Streuung), ist demnach die vierfache Auflösung notwendig.

```
class Identifikation
{
    ...
    public void ErzeugeTestsignal_4()
    {
        Zufall Rauschen = new Zufall(0.71036, 0.23906);

        for (int k=0; k<Abtastwerte; k++)
        {
            x[k] = Rauschen.Normal(0,1);
        }
    }
    ...
}
```

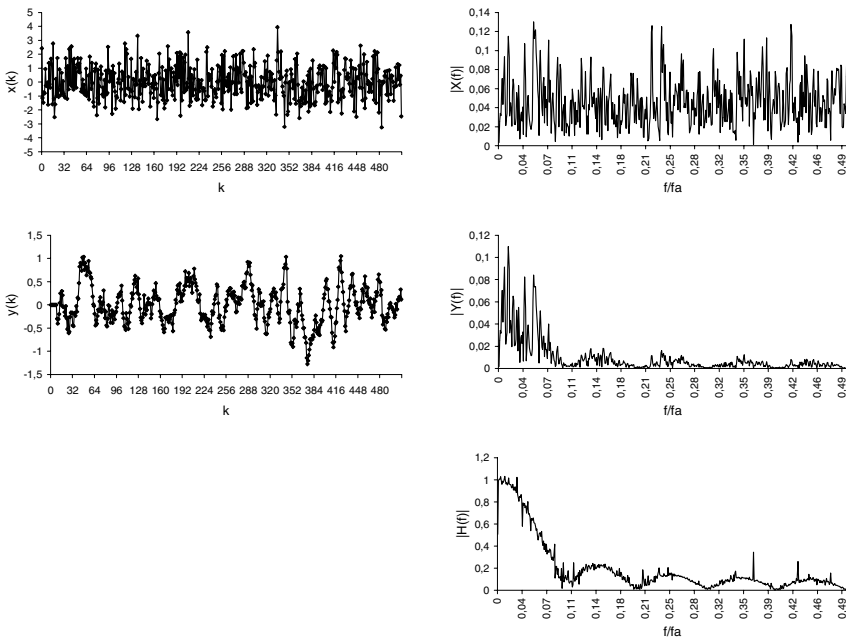


Abb. 1.49. Bestimmung der Übertragungsfunktion $|H(f)|$ mit Hilfe von Rauschen als Testsignal

$$x(k) = r(k)$$

$$y(k) = h(k) * r(k)$$

$$|H(f)| = \frac{DFT(h(k) * r(k))}{DFT(r(k))}$$

1.12.2 Identifikation im Zeitbereich

Möchte man die komplette Übertragungsfunktion bestimmen, dann kommt man um die zuvor beschriebene Frequenzanalyse nicht herum. Ist man dagegen z. B. nur an der Grenzfrequenz f_g interessiert, genügt schon eine Betrachtung im Zeitbereich. Zunächst machen wir aber einen kurzen Ausflug in die Analogtechnik. Wir stellen uns einen einfachen RC-Tiefpaß erster Ordnung vor, auf dessen Eingang wir einen Spannungssprung geben. Die Ausgangsspannung hat dann den folgenden Verlauf:

$$u_a(t) = u_0(1 - e^{-\frac{t}{\tau}})$$

Die Zeitkonstante $\tau = R \cdot C$ beschreibt, wie schnell die Ausgangsspannung ihren stationären Endwert erreicht, ist also ein Maß für die Dynamik des Systems. Systeme mit großen Zeitkonstanten reagieren recht träge, haben also eine geringe Dynamik. Umgekehrt stehen kleine Zeitkonstanten für kurze Reaktionszeiten, also eine hohe Dynamik. Allgemein sind die Zeitkonstanten von Übertragungssystemen umgekehrt proportional zum Frequenzbereich den diese verarbeiten können. Eine wesentliche Kenngröße zur Beschreibung von Tiefpässen im Zeitbereich ist deren Anstiegszeit t_a . Sie gibt an, in welcher Zeit das Ausgangssignal von 10% auf 90% des Endwertes ansteigt, wenn man eine Sprungfunktion an den Eingang legt.

$$t_a = t_{90\%} - t_{10\%} = \tau(\ln(0,9) - \ln(0,1)) \approx 2,2\tau \quad (1.53)$$

Somit ist die Anstiegszeit t_a etwas mehr als doppelt so groß, wie die Zeitkonstante τ unseres RC-Tiefpasses. Für die Grenzfrequenz $f_g = 1/2\pi\tau$ folgt daraus:

$$f_g \approx \frac{1}{3t_a} \quad (1.54)$$

Diese einfache Beziehung gestattet es, die Grenzfrequenz eines Übertragungssystems aus der Anstiegszeit des Ausgangssignals zu schätzen. Gln. (1.54) gilt näherungsweise auch für Systeme höherer Ordnung. Wir

wollen diese Erkenntnis nutzen, um die Grenzfrequenz eines rekursiven Tiefpaßfilters erster Ordnung zu bestimmen.

$$y(k) = x(k) + 0,5 \cdot y(k-1)$$

Bei der Implementierung rekursiver Systeme ist unbedingt der Rückgriff auf die alten Ausgangswerte zu berücksichtigen. Will man einen Absturz seines Computers vermeiden, dann muß die Schleife zur Berechnung, entsprechend der Filterordnung, mit einem Wert $k > 0$ starten.

```
class Identifikation
{
    ...
    public void Filterung()
    {
        // IIR-Tiefpassfilter 1. Ordnung
        for (int k=1; k<Abtastwerte; k++)
        {
            y[k] = x[k] + 0.5*y[k-1];
        }
    }
    ...
}
```

Da unseren synthetischen Daten, anders als die aufgezeichneten Daten aus einem echten Prozeß, natürlich keine reale Zeitbasis haben, ist es notwendig die einzelnen Zeitschritte auf die gewählte Abtastrate (im Beispiel $N = 128$) zu beziehen.

Die Werte für $t_{90\%}$ und $t_{10\%}$ können aus der Grafik abgelesen werden. Bequemer wäre es aber, eine deutlich höhere Abtastrate vorausgesetzt, sie direkt den tabellierten Ausgangswerten zu entnehmen. Im vorliegenden

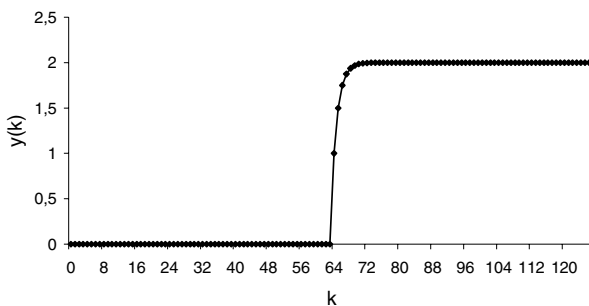


Abb. 1.50. Sprungantwort des Testsystems

Fall ergibt sich eine normierte Anstiegszeit von etwa $t_a = 3/128$. Eingesetzt in Gln. (1.54) erhalten wir damit eine auf die Grundschiwingung bezogene Grenzfrequenz von $f_g = 14,2$. Dieser Wert steht in recht guter Übereinstimmung mit dem nach einer Frequenztransformation (Fensterung nicht vergessen!) aus $|H(f)|$ abgelesenen Wert von 14,6.



<http://www.springer.com/978-3-540-20812-9>

Praktische Informationstechnik mit C#
Anwendungen und Grundlagen

Kluge, O.

2006, X, 304 S., Hardcover

ISBN: 978-3-540-20812-9