# Kapitel 3

# Interpolation von Punktmengen.

## 3.1 Definition des Problems.

Es seien die $x$- und die $y$-Koordinaten von $n$ Punkten gegeben. Die Zuordnung der Koordinaten sei *eindeutig* d.h. zu jedem $x$-Wert gehöre ein und nur ein $y$-Wert. Weiters soll angenommen werden, daß die $x$-Werte nach ihrer Größe geordnet vorliegen; sie müssen aber keineswegs äquidistant sein.

Eine wichtige Forderung an die Punktmenge ist mathematisch schwer zu definieren. Sie lautet: Die *Stützpunkte* $x_i \mid y_i$, $i = 1, \ldots, n$ sollen *einen funktionellen Zusammenhang zwischen $x$ und $y$ phänomenologisch befriedigend repräsentieren.* Die Abb.3.1 zeigt, wie das gemeint ist.
Geht man nun davon aus, daß die gegebene Punktmenge im Intervall $[x_1, x_n]$ repräsentativ ist, so kann eine *Interpolationsfunktion* $I(x)$ wie folgt definiert werden:

- $I(x)$ soll exakt durch die Stützpunkte hindurchgehen. Es soll also gelten:
$$I(x_i) = y_i \quad \text{für} \quad i = 1, \ldots, n \tag{3.1}$$
- $I(x)$ soll die Stützpunkte *möglichst glatt* verbinden. (3.2)

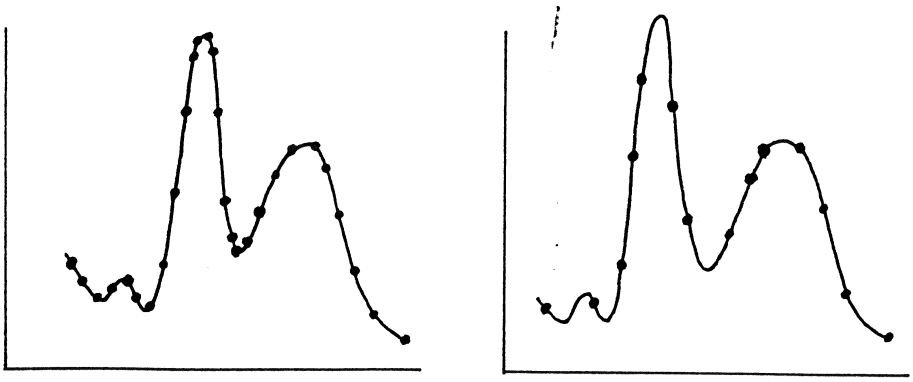


Abbildung 3.1: Punktmenge repräsentativ nicht- repräsentativ

Das folgende aus [13], S.127: Der Konstruktionsweg für die Interpolationsfunktion besteht darin, eine endliche Menge Funktionen $\varphi_k(x)$ zugrunde zu legen, die über dem Intervall $[x_1, x_n]$ linear unabhängig sind, und die Interpolationsfunktion als *Linearkombination* dieser Funktionen anzusetzen, also

$$I(x) = \sum_{k=1}^{n} c_k \varphi_k(x) \quad .$$

Typische Entwicklungsfunktionen für $I(x)$ sind

- Potenzfunktionen: $1 \; ; \; x \; ; \; x^2 \; ; \; x^3 \; ; \; \ldots$
- Exponentialfunktionen: $1 \; ; \; e^{\pm iax} \; ; \; e^{\pm 2iax} \; ; \; e^{\pm 3iax} \; ; \; \ldots$

## 3.2 Interpolation durch Potenzfunktionen.

Sehr häufig angewendet wird die *polynomiale Interpolation*, d.h. als Ansatz dient ein Polynom $(n-1)$-ten Grades mit den (vorerst) allgemeinen Koeffizienten $c_k$:

$$I(x) = \sum_{k=1}^{n} c_k x^{k-1} \tag{3.3}$$

Die $n$ Koeffizienten $c_k$ können nun dadurch ermittelt werden, daß man den Ansatz 3.3 in die Glg. 3.1 einsetzt:

$$I(x_i) = \sum_{k=1}^{n} c_k x_i^{k-1} = y_i \quad \text{für} \quad i = 1, \ldots, n$$

Dies stellt ein lineares inhomogenes Gleichungssystem für die Koeffizienten $c_1 \cdots c_n$ dar. Die Lösung eines solchen Problems ist *eindeutig* d.h. es gibt nur *ein* Polynom vom Grade $n-1$, das durch alle Stützpunkte exakt hindurchgeht. Damit ist die Bedingung (3.1) erfüllt.

Lagrange hat eine Formel angegeben, die es erlaubt, ein derartiges *Interpolationspolynom* in geschlossener Form anzuschreiben:

$$I(x) = \sum_{k=1}^{n} y_k \prod_{i=1(i\neq k)}^{n} \frac{(x - x_i)}{(x_k - x_i)} \tag{3.4}$$

Die numerische Berechnung von Werten des Interpolationspolynoms kann direkt aus der Gleichung (3.4) erfolgen. In der einschlägigen Literatur findet man jedoch zahlreiche Algorithmen für die Auswertung von (3.4), die für den Computer besser geeignet sind, wie z.B. den *Neville'schen Algorithmus* ([9], S.80ff) oder den *Formalismus von Newton* ([2], S.135ff, Programme S.345ff). In diesem Skriptum wird auf keines dieser Bibliotheksprogramme näher eingegangen, weil sie in der Praxis von nur beschränkter Bedeutung sind. Die Begründung dafür folgt im folgenden Beispiel (aus [14], S.2ff):

Gegeben seien 11 Punkte der Funktion

$$f(x) = \frac{1}{1 + x^2} \tag{3.5}$$

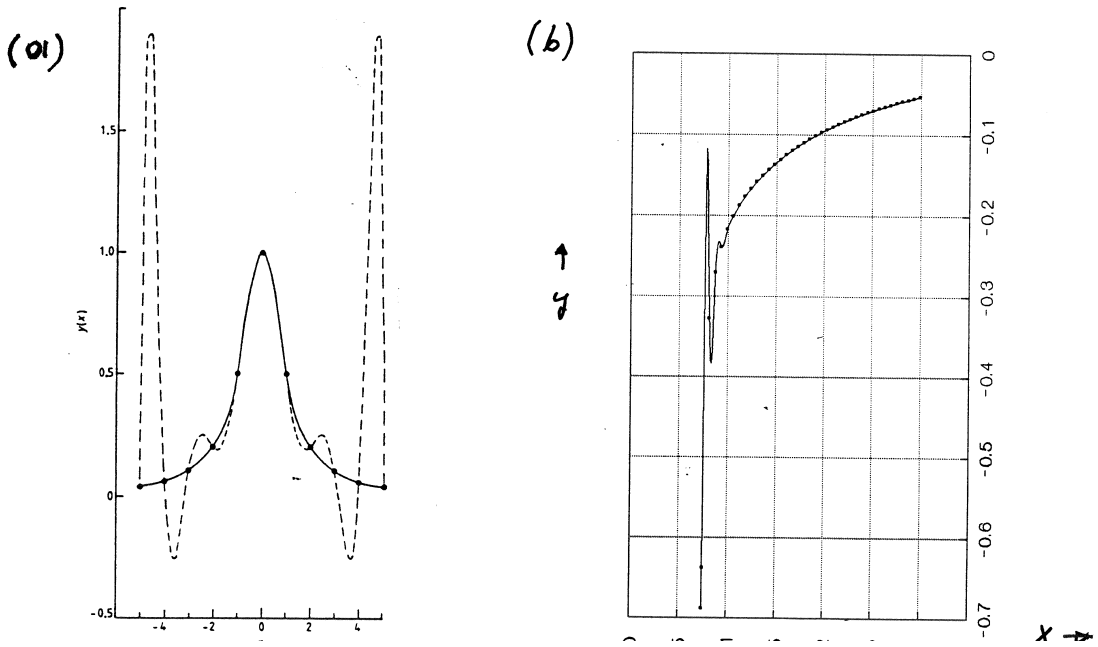



Abbildung 3.2: (a) Polynomiale Interpolation und stückweise Spline-Interpolation der Funktion (3.5); (b) Oszillationen einer Spline-Interpolation.

für die Argumente $x = -5, -4, \ldots, 4, 5$. Legt man durch diese 11 Stützpunkte das entsprechende Interpolationspolynom 10.ten Grades, so erhält man als Ergebnis die strichlierte Kurve in Abb. 3.2 (a).
Damit ist das Problem evident: das Polynom beschreibt die zu interpolierende Funktion nur im Bereich $x \approx 0$ gut. An den Rändern des Interpolationsintervalles kommt es hingegen zu massiven Verfahrensfehlern. Die Ursache dafür ist die Neigung von Interpolationspolynomen, zwischen den Stützpunkten zu oszillieren. Da diese Oszillationen mit steigendem Grad des Polynoms i.a. zunehmen, sind die gemäß (3.4) bestimmten Interpolationskurven in vielen Fällen in eindeutigem Widerspruch zur 'Glätte-Bedingung' (3.2) und somit unbrauchbar!

Im folgenden wird nun gezeigt, daß es meist zielführender ist, eine gegebene Punktmenge *stückweise* unter Verwendung von Polynomen niedriger Grade zu interpolieren.

### 3.2.1 Stückweise Interpolation mittels kubischer Splines.

Ausgangspunkt für die Spline-Interpolation sind Polynome vom Grad 3:

$$P^i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \quad , \tag{3.6}$$

wobei der Index $i$ bedeutet, daß das entsprechende Polynom nur im Intervall $[x_i, x_{i+1}]$ betrachtet wird. Die Interpolationskurve $I(x)$ besteht also (im Falle

von $n$ Stützpunkten) aus insgesamt $n-1$ Polynom-Stücken:

$$I(x) = \begin{cases} P^1(x) & \text{für} \quad x \in [x_1, x_2] \\ P^2(x) & \text{für} \quad x \in [x_2, x_3] \\ & \cdot \\ & \cdot \\ P^{n-1}(x) & \text{für} \quad x \in [x_{n-1}, x_n] \end{cases}$$

Die insgesamt $4(n-1)$ Polynomkoeffizienten $a_i$, $b_i$, $c_i$ und $d_i$ werden nun so ermittelt,daß

- $P^i(x)$ durch die Stützpunkte $y_i$ bzw. $y_{i+1}$ hindurchgeht.
- $P^i(x)$ am Punkt $x_{i+1}$ *dieselbe erste und zweite Ableitung* hat wie das rechts anschließende Polynomstück $P^{i+1}(x)$.

Die erste dieser beiden Bedingungen definiert eine Interpolationskurve, die zweite macht aus $I(x)$ *eine kubische Spline-Interpolation.*

Im Prinzip können zur Spline-Interpolation auch Polynome höherer Grade herangezogen werden. Da aber die kubischen Splines am weitaus häufigsten verwendet werden, soll hier nur von diesen die Rede sein (Informationen, Formeln und Programme zu Splines 5. Grades s. z.B. [2],S. 154ff bzw. S.358ff).

Für die kubischen Splines ergeben sich also die folgenden Bedingungen:

$$\begin{aligned} P^i(x_i) &= y_i & & \\ & & & i = 1, \ldots, n-1 \\ P^i(x_{i+1}) &= y_{i+1} & & \\ & \cdot & & \\ \tfrac{d}{dx} P^i(x_{i+1}) &= \tfrac{d}{dx} P^{i+1}(x_{i+1}) & & \\ & & & i = 1, \ldots, n-2 \\ \tfrac{d^2}{dx^2} P^i(x_{i+1}) &= \tfrac{d^2}{dx^2} P^{i+1}(x_{i+1}) & & \end{aligned} \tag{3.7}$$

Das sind $4n-6$ Gleichungen für die insgesamt $4n-4$ Polynomkoeffizienten. Es bedarf also zweier weiterer Gleichungen, um das System eindeutig lösbar zu machen. Diese beiden zusätzlichen Bedingungen sind im Prinzip beliebig. *Häufig fordert man, daß die 2. Ableitungen der Interpolationsfunktion am Beginn und am Ende des Interpolationsintervalls verschwinden*:

$$\frac{d^2}{dx^2} P^1(x_1) = 0 \qquad \frac{d^2}{dx^2} P^{n-1}(x_n) = 0 \tag{3.8}$$

Die so erhaltene Interpolationskurve ist stetig bzgl. des Funktionswertes sowie der ersten und der zweiten Ableitung und gewährleistet eine möglichst glatte Verbindung der gegebenen Stützpunkte. In der Literatur werden Splines mit den Nebenbedingungen (3.8) *natürliche* kubische Splines (*natural cubic splines*)genannt. In vielen Fällen ist es jedoch wünschenswert, diese Nebenbedingungen den Gegebenheiten des konkreten Problems anzupassen. Solche Variationen zum Thema kubische Splines findet man in [2], S.146ff, ausführlich beschrieben. Hier soll nur kurz auf eine Form der Nebenbedingungen eingegangen werden, die in der Literatur (z. B. bei [21], S. 450f)

oft den natürlichen Splines vorgezogen wird: die kubische Splinefunktion mit *not-a-knot*-Bedingung. Diese Bedingung mit

$$P^1(x) = P^2(x) \quad \text{für} \quad x_1 \leq x \leq x_3$$

und

$$P^{n-2}(x) = P^{n-1}(x) \quad \text{für} \quad x_{n-2} \leq x \leq x_n$$

besagt ([2], S.147), daß das erste und zweite Subintervall durch dasselbe Polynom und auch das vorletzte und letzte Subintervall durch dasselbe Polynom approximiert werden. Damit sind $x_2$ und $x_{n-1}$ keine 'echten' Knoten der Splinefunktion mehr ('not-a-knot').

Die folgenden Ausführungen sind nun wieder für *natürliche* kubische Splines: Im Prinzip könnte man aus dem Gleichungssystem (3.7) und (3.8) alle Spline-Koeffizienten berechnen. In der Praxis geht man etwas anders vor, nämlich über die folgenden Gleichungen, die eindeutig aus (3.7) und (3.8) resultieren. Die entsprechende Ableitung ist nicht schwierig, aber sehr langwierig, und soll daher hier wegbleiben:

$$\begin{aligned} a_i &= y_i \quad i = 1, \ldots, n \\ c_1 &= 0 \\ c_n &= 0 \end{aligned} \tag{3.9}$$

$$h_{i-1}c_{i-1} + 2c_i(h_{i-1} + h_i) + h_i c_{i+1} = \frac{3r_i}{h_i} - \frac{3r_{i-1}}{h_{i-1}} \quad i = 2, \ldots, n-1 \tag{3.10}$$

mit $h_i = x_{i+1} - x_i$ und $r_i = y_{i+1} - y_i$.

(3.10) bedeutet ein lineares inhomogenes Gleichungssystem $(n-2)$-ter Ordnung für die Unbekannten $c_2 \cdots c_{n-1}$, wobei die Koeffizientenmatrix *tridiagonal und symmetrisch* ist.

Aus der Kenntnis der $a_i$ und $c_i$ ergeben sich für die übrigen Spline-Koeffizienten die Bestimmungsgleichungen

$$\begin{aligned} b_i &= \tfrac{1}{h_i} r_i - \tfrac{h_i}{3}(c_{i+1} + 2c_i) \\ & \qquad\qquad i = 1, \ldots, n-1 \\ d_i &= \tfrac{1}{3h_i}(c_{i+1} - c_i) \end{aligned} \tag{3.11}$$

### 3.2.2 Das Programm SPLINE.

Das Programm SPLINE berechnet die Koeffizienten $a_i \ldots d_i$ natürlicher kubischer Splines für eine gegebene Menge von Stützpunkten $(x_i \mid y_i)$, $i = 1, \ldots, n$, wobei die $x_i$ monoton ansteigend geordnet sein müssen.

INPUT-Parameter:

**N:** Anzahl der Stützpunkte.

**X( ),Y( ):** Koordinaten der Stützpunkte.

OUTPUT-Parameter:

**A( ),B( ),C( ),D( ):** eindimensionale Felder mit den Spline-Koeffizienten $a_i$, $b_i$, $c_i$ und $d_i$.

Interne Felder:

**H( ),R( ),HD( ),UD( ),OD( ),INHOM( ),LOES( ):** .

Programmstruktur:

1. Check, ob die $x_i$ monoton ansteigen. Wenn nicht, Fehlermitteilung und Return.
2. Berechnung der Koeffizienten des Gleichungssystems (3.10) und Lösung dieses Systems mittels TRID (s. Abschnitt 2.6.1).
3. Berechnung der Spline-Koeffizienten gemäß den Formeln (3.9) und (3.11).

**Struktogramm 9** — SPLINE(N,X,Y,A,B,C,D)

<table>
<tr><td>I=1(1)N-1<br><table><tr><td>X(I) < X(I+1)<br>Y / N</td></tr><tr><td>H(I):=X(I+1)-X(I)<br>R(I):=Y(I+1)-Y(I)<br><br>(return 'Reihenfolge der x inkorrekt')</td></tr></table></td></tr>
<tr><td>J=1(1)N-2<br><table><tr><td>HD(J):=2.0*(H(J)+H(J+1))<br>UD(J):=H(J)<br>OD(J):=H(J+1)<br>INHOM(J):=3.0*(R(J+1)/H(J+1)-R(J)/H(J))</td></tr></table></td></tr>
<tr><td>TRID(UH,HD,OD,INHOM,N-2,LOES)</td></tr>
<tr><td>J=1(1)N-2<br><table><tr><td>C(J+1):=LOES(J)</td></tr></table></td></tr>
<tr><td>C(1):=0.0<br>C(N):=0.0</td></tr>
<tr><td>I=1(1)N-1<br><table><tr><td>A(I):=Y(I)<br>B(I):=R(I)/H(I)-H(I)*(C(I+1)+2.0*C(I))/3.0<br>D(I):=(C(I+1)-C(I))/3.0/H(I)</td></tr></table></td></tr>
<tr><td>(return)</td></tr>
</table>

### 3.2.3 Das Programm SPLVAL.

Das Programm SPLVAL (SPLine VALue) berechnet den Interpolationswert natürlicher kubischer Splines für ein beliebiges Argument $x \in [x_1, x_n]$.

**N:** Anzahl der Stützpunkte.

**X( ):** $x$-Koordinaten der Stützpunkte.

**A( ),B( ),C( ),D( ):** Koeffizienten kubischer Splines, wie sie z.B. vom Programm SPLINE berechnet werden.

**XX:** Argument, für welches die Interpolationsfunktion ausgewertet werden soll.

OUTPUT-Parameter:

**YY:** Interpolationswert.

**YYD:** erste Ableitung der Interpolationsfunktion an der Stelle XX.

**YYDD:** zweite Ableitung der Interpolationsfunktion an der Stelle XX.

**FEHLER:** diese Integer-Variable zur Fehlerdiagnostik hat nach der Exekution von SPLVAL den Wert '1', wenn XX außerhalb des Interpolationsbereiches liegt, sonst den Wert '0'.

Im Fall FEHLER=1 kehrt das Programm SPLVAL **ohne brauchbare Werte für YY, YYD und YYDD** ins aufrufende Programm zurück.

Anmerkung:

Der erste Teil von SPLVAL, in dem der Index J jenes Intervalls bestimmt wird, zu welchem XX gehört, ist ein Zitat aus dem Programm LOCATE ([9], S. 111).

**Struktogramm 10** — SPLVAL(N,X,A,B,C,D,XX,YY,YYD,YYDD,FEHLER)

EPS:=0.00001

XX < (X(1)-EPS) or XX > (X(N)+EPS)

Y:

(error 'Argument nicht im Interpolationsbereich')
FEHLER:=1
(return)

N:

.....

FEHLER:=0
JL:=0
JU:=N+1

JU-JL > 1

JM:=(JU+JL)/2

XX > X(JM)

Y:

JL:=JM

N:

JU:=JM

J:=JL

J=0

Y:

J:=1

N:

.....

J=N

Y:

J:=N-1

N:

.....

Z:=XX-X(J)
YY:=((D(J)*Z+C(J))*Z+B(J))*Z+A(J)
YYD:=(3.0*D(J)*Z+2.0*C(J))*Z+B(J)
YYDD:=6.0*D(J)*Z+2.0*C(J)

(return)

### 3.2.4 Beispiele für die kubische Spline-Interpolation.

**Beispiel 1: Demonstration einer Interpolation der fünf Stützpunkte**

| x | y |
|---|---|
| 1.0 | 0.2 |
| 1.6 | −0.1 |
| 1.9 | −0.6 |
| 2.3 | 0.0 |
| 2.7 | 0.5 |

mittels natürlicher kubischer Splines.

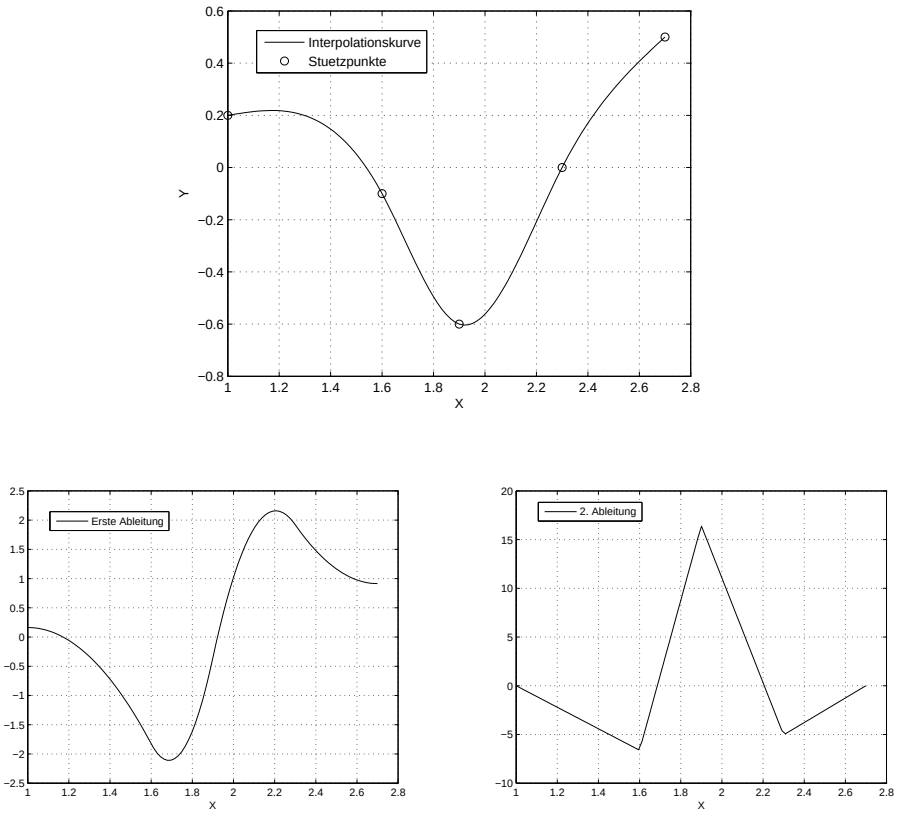



Abbildung 3.3: Demonstration einer Spline-Interpolation (Beispiel 1).

SPLINE liefert die folgenden Koeffizienten:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | 0.2000 | 0.1628 | 0.0000 | -1.8410 |
| 2 | -0.1000 | -1.8256 | -3.3139 | 12.8117 |
| 3 | -0.6000 | -0.3547 | 8.2167 | -8.9497 |
| 4 | 0.0000 | 1.9228 | -2.5229 | 2.1024 |

Die entsprechende Interpolationsfunktion $I(x)$ sowie die erste und zweite Ableitung von $I(x)$ sind in der Abbildung 3.3 dargestellt.

**Beispiel 2:**

Die Leistungsfähigkeit einer Interpolation mittels kubischer Splines, insbesondere was die Glattheit der Interpolationskurve betrifft, kann auch der Abb.3.1 entnommen werden (ausgezogene Kurve).

**Beispiel 3: Effektives Kernpotential eines Kalium-Atoms.**

Das elektrische Potential in einem Atom kann in der sogenannten 'Hartree-Näherung' in der Form

$$V(\mathbf{r}) = \frac{Ze}{4\pi\epsilon_o r} - \frac{e}{4\pi\epsilon_o}\int d\mathbf{z}\frac{\rho_e(\mathbf{z})}{\mid \mathbf{r}-\mathbf{z} \mid} \tag{3.12}$$

geschrieben werden, wobei der erste Term das Kernpotential ($Z$ = Kernladungszahl) und der zweite Term das Potential der den Kern umhüllenden Elektronen beschreibt. $\rho_e(|\mathbf{z}|)$ bedeutet die radialsymmetrische Näherung der Wahrscheinlichkeitsdichte der Elektronen an einem bestimmten Aufpunkt $\mathbf{z}$. Gleichung (3.12) läßt sich in der Form

$$V(r) = \frac{e}{4\pi\epsilon_o r}\underbrace{\left(Z - 4\pi\int_{z=0}^{r} dz z^2\rho_e(z) - 4\pi r\int_{z=r}^{\infty} dz z\rho_e(z)\right)}_{Z_{eff}(r)} \tag{3.13}$$

darstellen, wobei der Klammerausdruck die vom Radius $r$ abhängige *effektive Kernladungszahl* darstellt.

Im folgenden soll nun $Z_{eff}(r)$ für das Kalium-Atom berechnet werden (Z=19), wobei $z^2\rho_e(z)$ in Form einer Tabelle[1] gegeben ist:

```
Radiale Elektronendichte fuer das Kalium-Atom.
C. Froese-Fischer, Atomic Data 4, 301-399 (1972).

Erste Spalte der Tabelle: Radius z in Bohr
Zweite Spalte der Tabelle: z*z*rho(z)
mit rho(z)=Wahrscheinlichkeitsdichte der Elektronen.

Zahl der Tabellenwerte:
98

 0.0000     0.0000000E+00
 0.0001     0.4498521E-04
 0.0002     0.1799408E-03
 0.0004     0.7158627E-03
 0.0006     0.1597434E-02
 0.0008     0.2819985E-02
 0.0010     0.4370591E-02
   .
   .
10.0000     0.1369750E-02
11.0000     0.6779086E-03
12.0000     0.3228840E-03
```

[1]Die zur Erstellung dieser Tabelle erforderlichen Elektronen-Wellenfunktionen stammen aus: C. Froese Fischer, Atomic Data **4**,301-399 (1972). Wenn Sie dieses Beispiel rechnen wollen, können Sie diese Daten von der Web-Seite dieser Vorlesung unter *testdaten/kalium.dat* herunterladen.

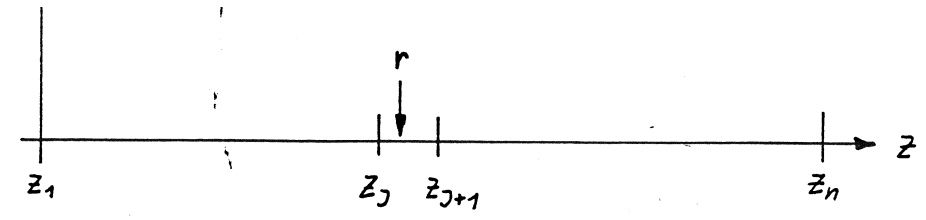



Abbildung 3.4: Integrationsbereiche der Integrale in Glg.3.12.

```
13.0000      0.1491913E-03
14.0000      0.6738352E-04
15.0000      0.2994823E-04
16.0000      0.1283439E-04
17.0000      0.5481809E-05
18.0000      0.2320359E-05
```

Die numerische Auswertung der beiden in (3.13) enthaltenen Integrale kann nun so erfolgen, daß man die Spline-Koeffizienten der Stützpunkte $[r_i \mid r_i^2\rho_e(r_i)]$ (erstes Integral) bzw. der Stützpunkte $[r_i \mid r_i\rho_e(r_i)]$ (zweites Integral berechnet. Bezeichnet man die zur ersten bzw. zweiten Funktion gehörenden Spline-Koeffizienten mit $A \ldots D$ bzw. $a \ldots d$, so ergibt sich:

$$z^2\rho_e(z) \approx A_i + B_i(z - z_i) + C_i(z - z_i)^2 + D_i(z - z_i)^3$$

beziehungsweise

$$z\rho_e(z) \approx a_i + b_i(z - z_i) + c_i(z - z_i)^2 + d_i(z - z_i)^3$$

für $z \in [z_i, z_{i+1}]$ und $i = 1, \ldots, n-1$. Mit diesen Approximationen geht man in die beiden Integrale, wobei zu berücksichtigen ist, daß der Integrationsbereich der ersten Integrals von 0 bis $r$ und der des zweiten Integrals von r bis $\infty$ geht ($\infty$ wird für die numerische Rechnung mit dem Argument des letzten Tabellenwertes gleichgesetzt). Liegt $r$ im $J$-ten Intervall der Wertetabelle, so umfaßt der erste Integrationsbereich die ersten $J-1$ Intervalle sowie einen Teil des $J$-ten Intervalls, der zweite Integrationsbereich den Rest des $J$-ten Intervalls sowie alle restlichen Intervalle (siehe Abb.3.4).
Man erhält somit für das erste Integral näherungsweise

$$\sum_{i=1}^{J-1} \int_{z=z_i}^{z_{i+1}} dz \left[A_i + B_i(z - z_i) + C_i(z - z_i)^2 + D_i(z - z_i)^3\right] + \\ + \int_{z=z_J}^{r} dz \left[A_J + B_J(z - z_J) + C_J(z - z_J)^2 + D_J(z - z_J)^3\right]$$

und für das zweite Integral

$$\int_{z=r}^{z_{J+1}} dz \left[a_J + b_J(z - z_J) + c_J(z - z_J)^2 + d_J(z - z_J)^3\right] + \\ + \sum_{i=J+1}^{n-1} \int_{z=z_i}^{z_{i+1}} dz \left[a_i + b_i(z - z_i) + c_i(z - z_i)^2 + d_i(z - z_i)^3\right]$$

Diese Integrale können leicht ausgewertet werden, und man erhält auf diese Weise einen einfachen Algorithmus für die Berechnung der effektiven Kernladungszahl für einen beliebigen Kernabstand $r$ im Intervall $r \in [0, z_n]$:

| r[Bohr] | Zeff |
|---|---|
| 0.00 | 18.9999 |
| 0.25 | 9.1943 |
| 0.50 | 5.8019 |
| 0.75 | 4.0352 |
| 1.00 | 2.7497 |
| 1.50 | 1.3052 |
| 2.00 | 0.7475 |
| 2.50 | 0.5122 |
| 3.00 | 0.3805 |
| 4.00 | 0.2152 |
| 5.00 | 0.1152 |
| 7.00 | 0.0279 |
| 10.00 | 0.0025 |
| 14.00 | 0.0001 |
| 18.00 | 0.0001 |

Dieses Ergebnis ist physikalisch leicht zu interpretieren: in großer Kernnähe wirkt die Ladung des Kalium-Kerns mit Ihren 19 Elementarladungen ungestört. Entfernt man sich jedoch vom Kern, so kompensieren die Hüllenelektronen sukzessive die Kernladung. In einem größeren Abstand ist diese 'Abschirmung' der Kernladung total, und das Atom bietet sich als neutrales Gebilde dar.

**Beispiel 4:**

Auch die Interpolation mit kubischen Splines ist kein fehlerfreies Verfahren. Wie die Abb. 3.2 (b) zeigt, kann es bei ungünstigen Stützpunkt-Verteilungen auch beim *splining* zu Oszillationen kommen.

### 3.2.5 Weitere Stichworte zum Thema: Interpolation.

Selbstverständlich gäbe es zum Thema *Interpolation* im allgemeinen und *Spline-Interpolation* im besonderen noch eine Menge Interessantes zu sagen. Ich muß mich hier auf einen knappen Themenkatalog beschränken. Einige dieser Themen werde ich in meiner ab dem SS 2002 angebotenen Lehrveranstaltung 'Ausgewählte Kapitel aus Numerische Methoden in der Physik' behandeln.

- Einen Überblick über die Verwendung von Spline-Funktionen samt Algorithmen finden Sie in dem Buch von H. Späth, *Spline-Algorithmen zur Konstruktion glatter Kurven und Flächen*, Oldenburg-Verlag München, 1978.

- Eine kurze Einführung in die Theorie mit vielen Formeln (ohne mathematische Beweise) bietet [2], S. 146ff. Dieses Buch (und die entsprechenden Werke für C [3] und PASCAL [4]) enthält auch eine Reihe von Programmlistings wie z.B.:
  *Interpolation mittels kubischer Splines mit wählbaren Randbedingungen; Interpolation mittels periodischer kubischer Splines; mittels parametrischer kubischer Splines; Interpolation mittels Splines fünften Grades; Polynomiale Ausgleichssplines dritten Grades; ...; zweidimensionale Polynom-Splines dritten Grades usw.*
  Es soll hier auch auf zwei neuere Publikationen von Frau Prof. Engeln-Müllges hingewiesen werden, nämlich:

  – [5] G. Engeln-Müllges und F. Reutter, *Numerik-Algorithmen mit Programmen in FORTRAN, C und Turbo Pascal*, VDI-Verlag, 1996.
  – [6] G. Engeln-Müllges and F. Uhlig, *Numerical Algorithms with C or FORTRAN*, Springer-Verlag, 1996.

  Die Programmcodes aus dem letzteren dieser Bücher [6] stehen Ihnen auf den PC's im Computerraum Physik unter

  ```
  /usr/local/numeric/num_alg_c/...      bzw.
  /usr/local/numeric/num_alg_f/...
  ```

  zur Verfügung.

- Zahlreiche Infomationen zum Thema Interpolation mit vielen Hinweisen auf entsprechende Software finden Sie auch im Buch von Überhuber [21], Kap. 9.

- Interpolation mittels rationaler Funktionen:

$$I(x) = \frac{p_0 + p_1 x + \ldots + p_\mu x^\mu}{q_0 + q_1 x + \ldots + q_\nu x^\nu}$$

  mit $\nu + \mu + 1$ gleich Anzahl der Stützpunkte ($q_0$ beliebig!).

  Literatur: [10], S. 111ff, [2], S. 176/S. 399ff.

- Mehrdimensionale Interpolation:

  Literatur: [10], S. 123ff, [2], S. 164ff/S. 392ff.

### 3.2.6 Software-Angebot

Zusätzlich zu den im vorigen Abschnitt erwähnten Programmen weise ich auf die folgenden Software-Produkte hin:

**PPPACK:** (p.d.) Paket, erhältlich über NETLIB, für die Berechnung und Manipulation stückweiser Polynome. Allerdings nur F77-Programme, z. B. die Programme *cubspl.f* und *ppvalu.f*, die im wesentlichen die Funktionen der Programme SPLINE und SPLVAL in diesem Skriptum haben, jedoch für wählbare Randbedingungen (inklusive der not-a-knotBedingung).

**IMSL:** Diese kommerzielle Bibliothek[2] gibt es für Großrechner, Workstations und PSs in Fortran und C.
Sie enthält eine Reihe von Unterprogrammen, mit deren Hilfe kubische Splinefunktionen berechnet werden können bzw. Programme, mit welchen kubische Splinefunktionen an vorgegebener Stelle ausgewertet werden können:

IMSL/MATH-LIBRARY

| | |
|---|---|
| /csint | Splines mit *not-a-knot* Randbedg. |
| /csdec | Splines mit vom Benutzer gewählten Randbedg. |
| /cscon | *optisch glatte* kubische Splinefunktion. |
| /csval | Auswertung einer Splinefunktion |
| /csval | Auswertung und Ableitung einer Splinefunktion. |

**Mathematica:** Interpolation[table,InterpolationOrder → n]
InterpolatingFunction[range,table]

**MATLAB:** Einige einfach zu verwendende Funktionen gehören zur Grundausstattung von MATLAB, nämlich die ein- zwei- und dreidimensionalen polynomialen Interpolationsroutinen *interp1, interp2* und *interp3*.

Ein sehr umfangreiches Angebot bietet die 'Spline Toolbox' von MATLAB, über die Sie sich mittels der MATLAB-HELP-Funktion informieren können. Im folgenden finden Sie eine (unvollständige) Liste der angebotenen Funktionen sowie eine Kurzbeschreibung der Funktion *csape*:

[2]International Mathematical and Statistical Libraries, Visual Numerics Inc., Houston, Texas, USA).

**csape** - Cubic spline interpolation with various end-conditions
**csapi** - Cubic spline interpolant with not-a-knot end condition
**csaps** - Cubic smoothing spline
**cscvn** - `Natural' or periodic cubic spline curve
**getcurve** - Interactive creation of a cubic spline curve
**ppmak** - Put together a spline in ppform
**spap2** - Least squares spline approximation
**spapi** - Spline interpolation
**spaps** - Smoothing spline
**spcrv** - Spline curve from control points by uniform subdivision
**spmak** - Put together a spline in B-form
**rsmak** - Put together a rational spline in rBform
**rpmak** - Put together a rational spline in rpform

## csape

**Purpose**

Cubic spline interpolation with end conditions

**Syntax**

```
pp = csape(x,y)
pp = csape(x,y,conds,valconds)
```

**Description**

A cubic spline *s* (in ppform) with knot sequence x is constructed that satisfies `s(x(j)) = y(:,j)` for all `j`, as well as an additional *end condition* at the first and at the last data site, as specified by *conds* and *valconds*.

*conds* may be a *string* whose first character matches one of the following: `'complete'` or `'clamped'`, `'not-a-knot'`, `'periodic'`, `'second'`, `'variational'`, with the following meanings.

| | |
|---|---|
| `'complete'` | Match endslopes (as given in *valconds*, with default as under "default") |
| `'not-a-knot'` | Make second and second-last sites inactive knots (ignoring *valconds* if given) |
| `'periodic'` | Match first and second derivatives at first site with those at last site |
| `'second'` | Match end second derivatives (as given in *valconds*, with default [0 0], i.e., as in `'variational'`) |
| `'variational'` | Set end second derivatives equal to zero (ignoring *valconds* if given) |
| default | Match endslopes to the slope of the cubic that matches the first four data at the respective end (i.e., Lagrange) |

By giving *conds* as a 1-by-2 matrix instead, it is possible to specify *different* conditions at the two endpoints. Explicitly, $D^i s$ is given the value *valconds*(*j*) at the left (*j* = 1) respectively right (*j* = 2) endpoint in case conds(*j*) = *i*, *i* = 1,2. There are default values for *conds* and/or *valconds*.

## 3.3 Fourier-Analyse diskreter Daten

Gegeben sei eine Menge von $n$ äquidistant verteilten Stützpunkten

$$(x_j = x_0 + j\Delta | y_i) \qquad j = 0, 1, \ldots, n-1 \quad ,$$

wobei die $x$ reellwertig und die $y$ (i. a.) komplexwertig sind. Um die Formeln zu vereinfachen, wird im folgenden $x_0 = 0$ angenommen. Diese Punktmenge soll nun unter Verwendung der im Abschnitt 3.1 erwähnten exponentiellen Basisfunktionen interpoliert werden:

$$y(x) = \frac{1}{n} \sum_{k=0}^{n-1} Y_k \, \mathrm{e}^{-i\alpha k x} \, . \tag{3.14}$$

Gleichzeitig soll angenommen werden, daß die $n$ gegebenen Punkte Teil einer periodischen Funktion mit der Periode

$$P = \Delta \, n$$

sind.[3] Demzufolge muß die Interpolationsfunktion $I(x)$ ebenfalls bzgl. $P$ periodisch sein, d.h. es muß gelten:

$$I(x + P) = I(x) \, .$$

Wie man leicht zeigen kann, ist diese Bedingung äquivalent mit einer speziellen Wahl der bisher freien Konstanten $\alpha$:

$$\frac{1}{n} \sum_{k=0}^{n-1} Y_k \, \mathrm{e}^{-i\alpha k(x+P)} = \frac{1}{n} \sum_{k=0}^{n-1} Y_k \, \mathrm{e}^{-i\alpha k x} \, .$$

Dies ist offenbar der Fall, wenn gilt

$$\mathrm{e}^{-i\alpha k P} = \mathrm{e}^{-i\alpha k n \Delta} = 1 \, ,$$

was der Bedingung

$$\alpha = \frac{2\pi}{\Delta n}$$

entspricht. Es ergibt sich somit für die Interpolationsfunktion der Ausdruck

$$I(x) = \frac{1}{n} \sum_{k=0}^{n-1} Y_k \, \exp\left[ -i \frac{2\pi k x}{\Delta n} \right] \, . \tag{3.15}$$

Nach der Interpolationsbedingung (3.1) gilt nun

$$I(x_j) = \frac{1}{n} \sum_{k=0}^{n-1} Y_k \, \exp\left( -i \frac{2\pi k j}{n} \right) = y_j \quad (j = 0, 1, \ldots, n-1) \tag{3.16}$$

mit den i. a. komplexwertigen Fourierkoeffizienten $Y_k$.

[3] Es spielt dabei keine Rolle, ob die zu interpolierenden Funktionswerte *tatsächlich* eine periodische Funktion beschreiben!

Selbstverständlich könnten die $Y_k$ direkt aus dem linearen Gleichungssystem (3.16) ermittelt werden. Im Falle der Fourier-Entwicklung geht das aber sehr viel einfacher. Multipliziert man nämlich die Glg. (3.16) mit $\exp(i2\pi k'j/n)$ und summiert über den Index $j$, so ergibt sich

$$\sum_{j=0}^{n-1} y_j \exp\left(i\frac{2\pi k'j}{n}\right) = \frac{1}{n}\sum_{k=0}^{n-1} Y_k \underbrace{\sum_{j=0}^{n-1} \exp\left(i\frac{2\pi j}{n}(k'-k)\right)}_{(A)} . \qquad (3.17)$$

o.B.: der Ausdruck (A) hat im Falle äquidistanter Stützpunkte das einfache Ergebnis

$$(A) = n \cdot \delta_{k,k'} \qquad \text{'Kronecker-Delta':} \quad 1 \quad \text{für} \quad k = k' \quad \text{sonst Null.}$$

Die Ursache dafür ist, daß die Fourier-Entwicklungsfunktionen bzgl. einer äquidistanten Punktmenge *orthogonal* sind.
Somit erhält man aus Glg. (3.17) unmittelbar

$$\sum_{j=0}^{n-1} y_j \exp\left(i\frac{2\pi k'j}{n}\right) = \frac{1}{n}\sum_{k=0}^{n-1} Y_k\, n\, \delta_{k,k'} = Y_{k'} .$$

Es ergibt sich somit für die *Fourierkoeffizienten* $Y_k$ die einfache Bestimmungsgleichung

$$Y_k = \sum_{j=0}^{n-1} y_j \exp\left(i\frac{2\pi kj}{n}\right) \qquad (k = 0, 1, \ldots, n-1) . \qquad (3.18)$$

Man nennt die Auswertung (3.18), also die Berechnung der $Y_k$ aus den $y_j$, die Fourier-Transformation (FT) der $y_j$; die Auswertung von (3.16) heißt dementsprechend die inverse FT.

Wegen der komplexwertigen Funktionenbasis der Fourier-Summe sind die Fourierkoeffizienten $Y_k$ im allgemeinen ebenfalls komplexwertig. Ein in der Praxis wichtiger Sonderfall ist gegeben, wenn die Datenwerte $y_j$ reell sind. In diesem Fall haben die Fourierkoeffizienten die Eigenschaft[4]

$$Y_k = Y^{\star}_{n-k} \quad . \qquad (3.19)$$

### 3.3.1 Numerische Berechnung der Fourierkoeffizienten

Die numerische Auswertung von (3.18) ist ein Prozess der Ordnung $n^2$, d.h. es sind $n$ Summenterme für jeden der $n$ Koeffizienten zu berechnen. Dies bedeutet für große $n$ einen sehr zeitintensiven Rechenprozess.

In bezug auf die Rechenzeit bietet die sogenannte *Fast Fourier Transform FFT*, die auf einer rekursiven Berechnung der in (3.18) vorkommenden Summe beruht, einen wesentlichen Fortschritt. Dieser Algorithmus, der auf den

[4]Bitte selbst verifizieren.

Ergebnissen einer Arbeit von Danielson und Lanczos (1942) beruht, ist zwar im Prinzip einfach, seine geschickte Programmierung ist jedoch im Detail etwas trickreich. Er soll deshalb hier nur prinzipiell erläutert werden, und zwar für das Beispiel von $n = 8$ Stützpunkten. Spaltet man die Summe (3.18) in zwei Teilsummen auf, und zwar in der Form

$$Y_k = \sum_{j=0}^{7} y_j e^{i\cdot 2\pi kj/8} = \sum_{j=0(2)}^{6} y_j e^{i\cdot 2\pi kj/8} + \sum_{j=1(2)}^{7} y_j e^{i\cdot 2\pi kj/8} \quad ,$$

so erhält man durch geeignete Transformation der Summenindizes den Ausdruck

$$Y_k = \sum_{j=0}^{3} y_{2j} e^{i\cdot 2\pi kj/4} + e^{i\cdot 2\pi k/8} \sum_{j=0}^{3} y_{2j+1} e^{i\cdot 2\pi kj/4}$$

bzw. unter Verwendung der Abkürzung $W_m = \exp(i \cdot 2\pi/m)$

$$Y_k = \sum_{j=0}^{7} W_8^{j\cdot k} y_j = \underbrace{\sum_{j=0}^{3} W_4^{j\cdot k} y_{2j}}_{Y_k^0} + W_8^k \underbrace{\sum_{j=0}^{3} W_4^{j\cdot k} y_{2j+1}}_{Y_k^1}$$

Die beiden Teilsummen $Y_k^0$ und $Y_k^1$ können nun ihrerseits wieder zerlegt werden:

$$Y_k^0 = \underbrace{\sum_{j=0}^{1} W_2^{j\cdot k} y_{4j}}_{Y_k^{00}} + W_4^k \underbrace{\sum_{j=0}^{1} W_2^{j\cdot k} y_{4j+2}}_{Y_k^{01}}$$

$$Y_k^1 = \underbrace{\sum_{j=0}^{1} W_2^{j\cdot k} y_{4j+1}}_{Y_k^{10}} + W_4^k \underbrace{\sum_{j=0}^{1} W_2^{j\cdot k} y_{4j+3}}_{Y_k^{11}}$$

Die letzte Aufspaltung der vier Größen $Y_k^{00}$, $Y_k^{01}$, $Y_k^{10}$ und $Y_k^{11}$ lautet:

$$Y_k^{00} = y_0 + W_2^k y_4 \qquad Y_k^{01} = y_2 + W_2^k y_6$$

$$Y_k^{10} = y_1 + W_2^k y_5 \qquad Y_k^{11} = y_3 + W_2^k y_7$$

Es sind somit die Fourierkoeffizienten 'auf die Ebene der $y$-Werte reduziert'. Davon ausgehend, kann der gewünschte Koeffizient $Y_k$ schrittweise aufgebaut werden, wie dies schematisch in der Abbildung 3.5 dargestellt ist.

Die eben präsentierte Vorgangsweise funktioniert jedoch nur dann, wenn die bei dem Aufbau der Koeffizienten gemäß Abb. 3.5 berücksichtigten Stützpunkte von Schritt zu Schritt verdoppelt werden können, d.h. *wenn die Stützpunktanzahl $n$ eine Potenz von 2 ist* [5].

Wie ebenfalls aus Abb. 3.5 hervorgeht, müssen die gegebenen $y_j$-Werte ($j = 0, \ldots, n-1$) am Beginn der Kaskade (links in der Abb.) 'geeignet'

[5] Diese strenge Bedingung ist in einigen modernen FFT-Programmen (s. Abschnitt 3.4.5) *nicht mehr* gegeben.

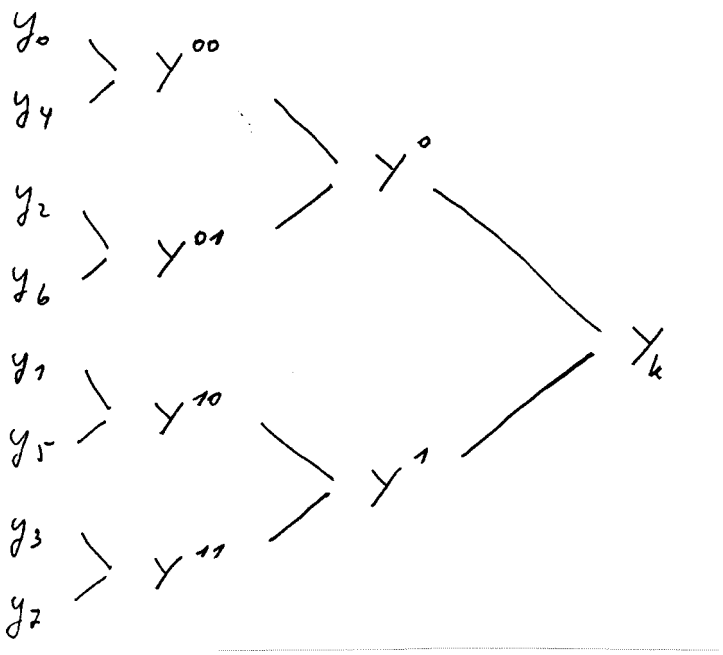



Abbildung 3.5: Grundprinzip der 'Fast Fourier Transform'.

geordnet werden. Es zeigt sich, daß diese Indexordnung sich einfach aus einer *Bitumkehr* der ursprünglichen Indizes in dualer Schreibweise ergibt:

| ursprüngliche Folge (Index) | Dual | BITUMKEHR | neue Folge (Index) |
|---|---|---|---|
| 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 |

Der große Vorteil der FFT gegenüber der direkten Auswertung von (3.18) liegt in der beträchtlichen Einsparung an Rechenzeit. Während nämlich bei der Berechnung der Fourierkoeffizienten gemäß (3.18) die Auswertung von $n^2$ Summentermen nötig ist, reduziert sich diese Zahl bei der FFT auf $n \ln_2(n)$:

| $n$ | $n^2$ | $n \ln_2(n)$ |
|---|---|---|
| 128 | 16384 | 896 |
| 1024 | 1048576 | 10240 |
| . | . | . |
| . | . | . |
| 1048576 | $\approx 10^{12}$ | $\approx 2.1 \cdot 10^7$ |

Bei sehr großen Datenmengen ist eine Fourier-Transformation mittels kleiner Computer (PC) überhaupt nur mittels der FFT möglich!

### 3.3.2 FFT-Programme in C, F90 und MATLAB

FFT-Programme sind häufig recht raffiniert programmiert. Es erscheint im Rahmen dieser LV nicht sinnvoll, auf alle Details der Algorithmen einzugehen. Aus diesem Grund werden im folgenden (ausnahmsweise) keine Struktogramme, sondern direkt FFT-Programmlistings in den Sprachen C, F90 und MATLAB präsentiert. Es handelt sich dabei um Programme zur schnellen Fourier-Transformation einer diskreten komplexwertigen Punktmenge mit $2^m$ Werten ($m$ = positive Integerzahl) und mit äquidistanten $x$-Komponenten.

#### Das C-Programm FOUR1

Quelle: [10], S. 507f.

INPUT-Parameter:

**DATA( ):** eindimensionales Feld vom Typ *double*, das die Real- und Imaginärteile der Größen enthält, die Fourier-transformiert werden sollen.

Die DATA-Werte müssen wie folgt abgespeichert sein:

```
RT{y0}          DATA(1)
IT{y0}          DATA(2)

RT{y1}          DATA(3)
IT{y1}          DATA(4)
   .
   .
RT{y(n-1)}      DATA(2n-1)
IT{y(n-1)}      DATA(2n)
```

DATA muß also mindestens $2n$ Werte aufnehmen können!

**NN:** Anzahl der $n$ (i. a. komplexen) Werte.

**ISIGN=+1:** Fourier-Transformation.

**ISIGN=−1:** inverse Fourier-Transformation (aber ohne den Normierungsfaktor $1/n$).

Die Variablen NN und ISIGN sind vom Typ *int*.

OUTPUT-Parameter:

**DATA( ):** Real-Feld mit den Real- und Imaginärteilen der Fourier-Transformierten.

```
#include <math.h>
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
```

```
void four1(double data[], int nn, int isign)
{
int n,mmax,m,j,istep,i;
double wtemp,wr,wpr,wpi,wi,theta;
double tempr,tempi;

n=nn << 1;
j=1;
for (i=1;i<n;i+=2) {
if (j > i) {
SWAP(data[j],data[i]);
SWAP(data[j+1],data[i+1]);
}
m=n >> 1;
while (m >= 2 && j > m) {
j -= m;
m >>= 1;
}
j += m;
}
mmax=2;
while (n > mmax) {
istep=mmax << 1;
theta=isign*(6.28318530717959/mmax);
wtemp=sin(0.5*theta);
wpr = -2.0*wtemp*wtemp;
wpi=sin(theta);
wr=1.0;
wi=0.0;
for (m=1;m<mmax;m+=2) {
for (i=m;i<=n;i+=istep) {
j=i+mmax;
tempr=wr*data[j]-wi*data[j+1];
tempi=wr*data[j+1]+wi*data[j];
data[j]=data[i]-tempr;
data[j+1]=data[i+1]-tempi;
data[i] += tempr;
data[i+1] += tempi;
}
wr=(wtemp=wr)*wpr-wi*wpi+wr;
wi=wi*wpr+wtemp*wpi+wi;
}
mmax=istep;
}
}
#undef SWAP
/* (C) Copr. 1986-92 Numerical Recipes Software +. */
```

**Das F90-Programm FFT**

Quelle: [23], S. 322f.

INPUT-Parameter:

**A():** eindimensionables Datenfeld vom Typ

```
COMPLEX(KIND=KIND(0.0d0))      ,
```

das die komplexwertigen Größen enthält, die Fourier-transformiert werden sollen.

**M:** $2**M$ ist die Zahl $n$ der (äquidistanten) Datenwerte.

**INV**$=+1$**:** Fourier-Transformation.

**INV**$\neq -1$**:** inverse Fourier-Transformation.

OUTPUT-Parameter:

**A( ):** COMPLEX-Feld mit den Real- und Imaginärteilen der Fourier-Transformierten.

```
      SUBROUTINE fft(a,m,inv)
! Paul L. DeVries, Department of Physics, Miami University

! This subroutine performs the Fast Fourier Transform by
! the method of Cooley and Tukey

!       start:                1965
!  last modified:             1993
!  Modifications H. Sormann:  2000
!

!  Parameter description by H. Sormann 2000
!  PARAMETERS:   a     vector of complex data which are to be
!                      Fourier-transformed
!                      this vector is VARIABLY dimensioned (see below)

!                      On return, a  contains the FT coefficients.
!
!                m     2**m  is the number of data points
!
!                inv equal 1        Fourier transform
!                inv not equal 1    INVERSE Fourier transform
!
      IMPLICIT NONE
```

```
      INTEGER                       :: m,inv,n,nd2,i,j,k,l,le,le1,ip
      COMPLEX(KIND=KIND(0.0d0))              :: u,w,t
      COMPLEX(KIND=KIND(0.0d0)),DIMENSION(*) :: a   ! * = var. dimension
      DOUBLE PRECISION              :: ang,pi
!      PARAMETER (pi=3.141592653589793d0)
      pi=4.d0*atan(1.d0)

      n=2**m
      nd2=n/2
      j=1
      DO i=1,n-1
        IF(i .lt. j)THEN
          t=a(j)
          a(j)=a(i)
          a(i)=t
        ENDIF
        k=nd2
100     IF(k .lt. j)THEN
          j=j-k
          k=k/2
          goto 100
        ENDIF
        j=j+k
      END DO
      le=1
      DO l=1,m
        le1=le
        le=le+le
        u=(1.d0,0.d0)
        ang=pi/dble(le1)
        w=cmplx(cos(ang), -sin(ang),KIND(0.d0))
        IF(inv .eq. 1)w=conjg(w)
        DO j=1,le1
          DO i=j,n,le
            ip=i+le1
            t=a(ip)*u
            a(ip)=a(i)-t
            a(i)=a(i)+t
          END DO
          u=u*w
        END DO
      END DO
      IF(inv .ne. 1)THEN
        DO i=1,n
          a(i)=a(i)/dble(n)
        END DO
```

```
      ENDIF
      END SUBROUTINE fft
```

### Das MATLAB-Programm fft

Es gibt eine interne MATLAB-Routine mit dem Funktionsnamen *fft.m*. Wenn Sie sich mittels *help fft* über diese Routine informieren, erhalten Sie den folgenden Text:

```
%FFT Discrete Fourier transform.
%   FFT(X) is the discrete Fourier transform (DFT) of vector X.  If the
%   length of X is a power of two, a fast radix-2 fast-Fourier
%   transform algorithm is used.  If the length of X is not a
%   power of two, a slower non-power-of-two algorithm is employed.
%   For matrices, the FFT operation is applied to each column.
%   For N-D arrays, the FFT operation operates on the first
%   non-singleton dimension.
%
%   FFT(X,N) is the N-point FFT, padded with zeros if X has less
%   than N points and truncated if it has more.
%
%   FFT(X,[],DIM) or FFT(X,N,DIM) applies the FFT operation across the
%   dimension DIM.
%
%   For length N input vector x, the DFT is a length N vector X,
%   with elements
%                    N
%      X(k) =       sum  x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k <= N.
%                   n=1
%   The inverse DFT (computed by IFFT) is given by
%                    N
%      x(n) = (1/N) sum  X(k)*exp( j*2*pi*(k-1)*(n-1)/N), 1 <= n <= N.
%                   k=1
%
%   The relationship between the DFT and the Fourier coeffs a and b in
%            N/2
%   x(n)=a0+ sum a(k)*cos(2*pi*k*t(n)/(N*dt))+b(k)*sin(2*pi*k*t(n)/(N*dt))
%            k=1
%   is
%      a0 = X(1)/N, a(k) = 2*real(X(k+1))/N, b(k) = -2*imag(X(k+1))/N,
%   x is a length N discrete signal sampled at times t with spacing dt.
%
%   See also IFFT, FFT2, IFFT2, FFTSHIFT.

%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.9 $  $Date: 1998/06/30 18:43:41 $

%   Built-in function.
```

Ein MATLAB-Programm, welches das Testproblem im Abschnitt 3.3.3 lösen soll, könnte etwa so aussehen:

```
%Eingabe der Testdaten von Skriptum:
n=8;
delta=1.0;
period=n*delta;

index=0:1:n-1;

werte=[0.7013+0.0437i -0.0724+0.5133i 0.0988-0.2688i 0.0715-0.1162i ...
       0.4013+0.1188i -0.0901-0.1408i -0.1263-0.0688i 0.2660-0.3813i];


yy=fft(werte);
```

Nun folgt eine zwar im Prinzip harmlose, aber lästige Komplikation: wenn man die Ergebnisse der Fourier-Transformation mittels dieses Matlab-Programms mit den entsprechenden Resultaten des C-Programms *four1.c* bzw. des Fortran-Programms *fft.f90* vergleicht, so erkennt man, daß *die Reihenfolge der Fourierkoeffizienten bei Matlab sich von den anderen Ergebnissen unterscheidet* (vgl. das Testbeispiel auf S. 83).

Das heißt nicht etwa, daß das Matlab-Programm falsch rechnet; die Ursache ist vielmehr, daß die Matlab-Version von einer zu Glg. (3.15) etwas verschiedenen - wenn auch mathematisch äquivalenten - Grundformel für die Interpolationsfunktion ausgeht.

Um nun alle daraus folgenden Unsicherheiten zu eliminieren, habe ich neue, sehr einfache Variationen der Matlab-Routinen *fft.m* bzw. *ifft.m* geschrieben, bei denen durch einfache Umordnungen der Fourierkoeffizienten dieses Problem nicht mehr auftritt:

```
function fourier = fft_sor(x)

% H. Sormann   20-9-2007
% last update  20-9-2007

% Um auch bei den Matlab-Programmen fft.m und ifft.m dieselbe
% Reihenfolge der Fourierkoeffizienten zu erhalten wie bei
% den im Skriptum praesentierten C-Programm (four1.c) bzw.
% F90-Programm (fft.f90), habe ich dieses einfache Programm
% geschrieben, bei welchem NACH der Anwendung von fft.m
% "automatisch" eine Umordnung der Matlab-Ergebnisse erfolgt.

% Wichtig ist, dass Sie die Programme fft_sor.m und ifft_sor.m
% paarweise verwenden, also nicht fft_sor.m mit ifft.m
% kombinieren und umgekehrt.

  n=length(x);
  d=fft(x);

  fourier(1)=d(1);
  for ind=2:n
    fourier(ind)=d(n+2-ind);
  end


function ifourier = ifft_sor(d)

% H. Sormann   20-9-2007
% last update  20-9-2007

% Um auch bei den Matlab-Programmen fft.m und ifft.m dieselbe
% Reihenfolge der Fourierkoeffizienten zu erhalten wie bei
% den im Skriptum praesentierten C-Programm (four1.c) bzw.
% F90-Programm (fft.f90), habe ich dieses einfache Programm
% geschrieben, bei welchem VOR der Anwendung von ifft.m
% "automatisch" eine Umordnung der Matlab-Ergebnisse erfolgt.

% Wichtig ist, dass Sie die Programme ifft_sor.m und fft_sor.m
% paarweise verwenden, also nicht ifft_sor.m mit fft.m
% kombinieren und umgekehrt.

  n=length(d);
  x(1)=d(1);
  for ind=2:n
    x(ind)=d(n+2-ind);
  end

  ifourier=ifft(x);
```

### 3.3.3 Ein Test für die FFT-Programme.

Dieses Beispiel beschreibt die Fourier-Transformation der folgenden Testdaten ($n = 8$, $\Delta = 1.$):

| j | RT{y(j)} | IT{y(j)} |
|---|---|---|
| 0 | 0.7013 | 0.0437 |
| 1 | -0.0724 | 0.5133 |
| 2 | 0.0988 | -0.2688 |
| 3 | 0.0715 | -0.1162 |
| 4 | 0.4013 | 0.1188 |
| 5 | -0.0901 | -0.1408 |
| 6 | -0.1263 | -0.0688 |
| 7 | 0.2660 | -0.3813 |

Die gegebenen Stützstellen repräsentieren somit eine periodische Funktion mit der Periode $P = 8$, und das Programm FOUR1 liefert die folgenden Fourierkoeffienten:

| k | RT{Y(k)} | IT{Y(k)} |
|---|---|---|
| 0 | 1.2501 | -0.3001 |
| 1 | 0.0001 | 0.3000 |
| 2 | 0.2601 | 0.0001 |
| 3 | -0.7000 | -0.7003 |
| 4 | 0.9001 | -0.0501 |
| 5 | 0.9999 | 0.0000 |
| 6 | 2.0001 | 1.0001 |
| 7 | 0.9000 | 0.0999 |

Beachten Sie, daß das *originale* Matlab-Programm *fft.m* eine veränderte Reihenfolge der Fourierkoeffizienten ausgibt (s. die Diskussion auf Seite 82f):

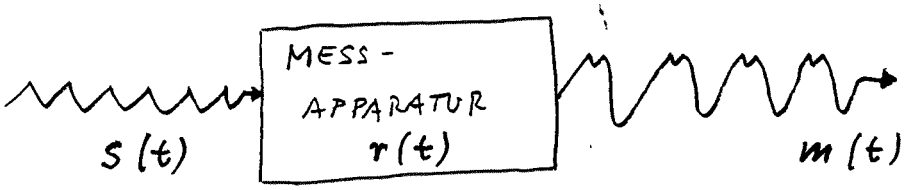
| k | RT{Y(k)} | IT{Y(k)} |
|---|---|---|
| 0 | 1.2501 | -0.3001 |
| 1 | 0.9000 | 0.0999 |
| 2 | 2.0001 | 1.0001 |
| 3 | 0.9999 | -0.0000 |
| 4 | 0.9001 | -0.0501 |
| 5 | -0.7000 | -0.7003 |
| 6 | 0.2601 | 0.0001 |
| 7 | 0.0001 | 0.3000 |

## 3.4 Wichtige Anwendungen der FT

In diesem Abschnitt sollen als Beispiele für die vielfältigen Verwendungsmöglichkeiten der FT in der Physik, Meßtechnik etc. drei Anwendungsbereiche erläutert werden.

### 3.4.1 Faltungsintegrale

Wenn irgendein Signal mittels einer Meßapparatur registriert wird, wird in den meisten Fällen das ursprüngliche Signal durch die Eigenschaften der Meßapparatur verändert:




Für die konkrete Auswertung der Meßdaten ist es i. a. sehr wichtig, diesen Einfluß der Apparatur im nachhinein so gut als möglich zu eliminieren.

o. B.: Nennt man die Signalfunktion $s(t)$ und die gemessene Funktion $m(t)$, so ist der Zusammenhang zwischen diesen Funktionen durch das sogenannte Faltungsintegral

$$m(t) = \int_{-\infty}^{+\infty} dt'\, r(t')\, s(t-t') \tag{3.20}$$

gegeben. Der Einfluß der Meßapparatur wird dabei durch die *Apparatefunktion* oder *Auflösungsfunktion (resolution function)* $r(t)$ beschrieben.

Es soll nun angenommen werden, daß die Funktionen $m(t)$ und $s(t)$ periodisch bzgl. der Periode $P$ sind. Außerdem soll angenommen werden, daß die Apparatefunktion auf diese Periode *beschränkt* sei, d.h. daß gilt:

$$r(t) = 0 \qquad \text{außerhalb} \quad [0, P]\,.$$

Unter diesen Umständen kann der Integrationsbereich in (3.20) auf $0 \le t' \le P$ beschränkt werden:

$$m(t) = \int_0^P dt'\, r(t')\, s(t-t')\,.$$

Dieses Integral kann nun mit Hilfe der Trapezformel (s. Kap. 6) näherungsweise ausgewertet werden. Mittels der im Integrationsintervall $[0, P]$ äquidistant verteilten Punkte

$$t_j = j\Delta \qquad \Delta = \frac{P}{n} \qquad (j = 0, 1, \ldots, n-1)$$

erhält man

$$m(t) \approx \Delta \sum_{j=0}^{n-1} r(t_j)\, s(t-t_j)\,.$$

Interessiert man sich nur für die Argumente

$$t_\ell = \ell\Delta\,,$$

so ergibt sich weiters

$$m(t_\ell) \approx \Delta \sum_{j=0}^{n-1} r(t_j)\, s(t_\ell - t_j)$$

bzw.

$$m_\ell \approx \Delta \sum_{j=0}^{n-1} r_j\, s_{\ell-j}\,. \tag{3.21}$$

Beachten Sie, daß bei der Auswertung der obigen Summe für $s$ negative Indizes vorkommen. Dies ist jedoch kein Problem, da wegen der (angenommenen) Periodizität von $r(t)$ gelten muß:

$$s_{-i} = s_{n-i}\,.$$

Nun kommt die FT ins Spiel! Setzt man nämlich für die drei in Glg. (3.21) vorkommenden Größen $m$, $r$ und $s$ die entsprechenden Fourierreihen gemäß (3.16) ein, so ergibt sich

$$\begin{aligned}
\frac{1}{n}\sum_{k=0}^{n-1} M_k\, \mathrm{e}^{-i2\pi k\ell/n} &\approx \Delta \sum_{j=0}^{n-1} \frac{1}{n}\sum_{k=0}^{n-1} S_k\, \mathrm{e}^{-i2\pi k(\ell-j)/n}\, \frac{1}{n}\sum_{k'=0}^{n-1} R_{k'}\, \mathrm{e}^{-i2\pi k' j/n} \\
&= \frac{\Delta}{n^2}\sum_{k=0}^{n-1}\sum_{k'=0}^{n-1} S_k\, R_{k'}\, \mathrm{e}^{-i2\pi k\ell/n} \underbrace{\sum_{j=0}^{n-1} \mathrm{e}^{i2\pi(k-k')j/n}}_{n\,\delta_{k,k'}} \\
&= \frac{\Delta}{n}\sum_{k=0}^{n-1} R_k\, S_k\, \mathrm{e}^{-i2\pi k\ell/n}\,.
\end{aligned}$$

Daraus folgt sofort die sehr wichtige Beziehung

$$M_k = \Delta\, R_k\, S_k\,. \tag{3.22}$$

D. h.: Abgesehen von der Konstanten $\Delta$ ist die FT des Faltungsergebnisses $m(t)$ einfach durch das Produkt der FT von $r$ und $s$ gegeben. Die Bedeutung von (3.22) für die Praxis ergibt sich daraus, daß man diese Gleichung ohne Probleme nach der *ungestörten* Signalfunktion auflösen kann:

$$S_k = \frac{1}{\Delta}\,\frac{M_k}{R_k}\,. \tag{3.23}$$

Kennt man also die gemessene Funktion $m(t)$ und die Apparatefunktion $r(t)$ im realen Raum, so kann man wie folgt vorgehen:

1. FT von $m(t)$ $\quad\rightarrow\quad$ $M_k$
2. FT von $r(t)$ $\quad\rightarrow\quad$ $R_k$
3. Berechnung von $S_k$ aus Glg. (3.23)
4. Inverse FT von $S_k$ $\quad\rightarrow\quad$ $s(t)$

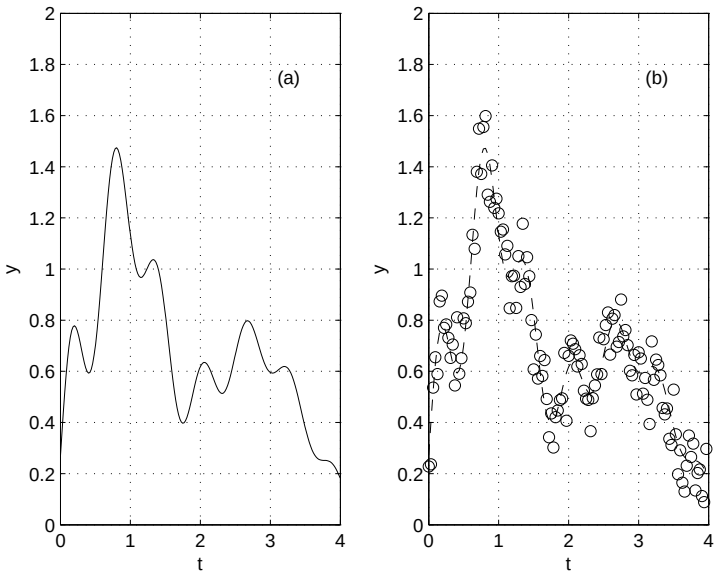



Abbildung 3.6: Demonstration einer Datenglättung mittels FT:
(a) ungestörtes Signal, (b) verrauschtes Signal.

### 3.4.2 Datenglättung

Zur Demonstration dieser Anwendung der FT zeigt die Abbildung 3.6 die Testfunktion

$$y(x) = \mathrm{e}^{-x/2}\left[2\mathrm{e}^{-2(x-1)^2} + 3\mathrm{e}^{-(x-3)^2} + \frac{1}{3}\sin(10x)\right]$$

im Intervall $0 \le x \le 4$. Abb. 3.6 (a) enthält die ungestörte Funktion, und in Abb. 3.6 (b) sind dem $y(x)$ statistische Fehler mit einer Standardabweichung von $\sigma = 0.1$ überlagert, wie sie bei Messungen unvermeidlich auftreten.
Die entsprechenden Beträge der Fourierkoeffizienten für $n$=128 sind in der Abb. 3.7 dargestellt. Wie man deutlich sieht, wirken sich die statistischen Fehler so aus, daß sich im 'mittleren Bereich' von $k$ (zum Unterschied von der glatten Kurve) von Null verschiedene Fourierkoeffizienten zeigen. Es liegt daher nahe, diese Koeffizienten dem *noise* zuzuschreiben und in geeigneter Weise zu unterdrücken.
Dies kann am einfachsten durch die *Regel von Dirichlet* geschehen, welche durch

$$\hat{Y}_k = Y_k \qquad \text{für} \quad 0 \le k < k_0 \quad \text{und} \quad n - k_0 < k < n$$
$$\hat{Y}_k = 0 \qquad \text{für} \quad k_0 \le k \le n - k_0$$

definiert ist, wobei der Index $k_0$ frei wählbar ist.

Es ist natürlich klar, daß die Qualität der *noise*-Unterdrückung entscheidend vom gewählten $k_0$ abhängt. In unseren Beispiel hat sich $k_0 = 9$ als sehr gute Wahl herausgestellt; allerdings muß zugegeben werden, daß die Kenntnis der *Idealkurve* - die natürlich im Praxisfall nicht gegeben ist - diese Wahl sehr erleichtert! Die hohe Qualität der Dirichlet-Glättung wird in der Abb. 3.8 demonstriert.

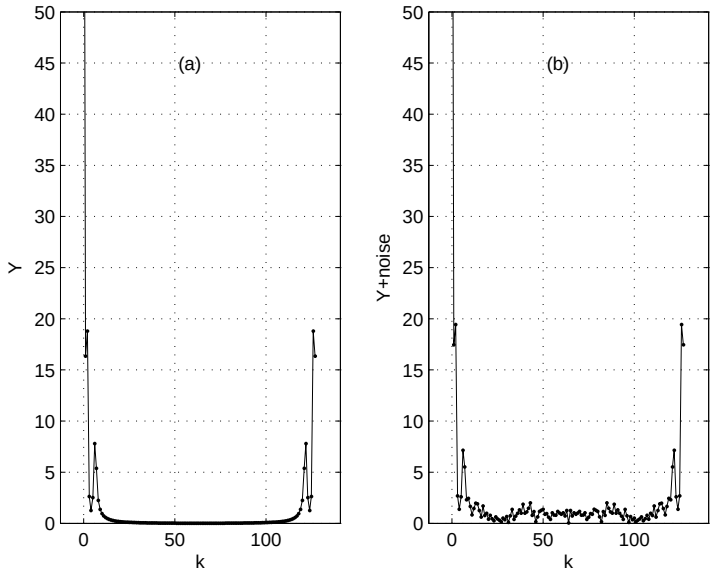



Abbildung 3.7: Demonstration einer Datenglättung mittels FT:
(a) FT des ungestörten Signals, (b) FT des verrauschten Signals.

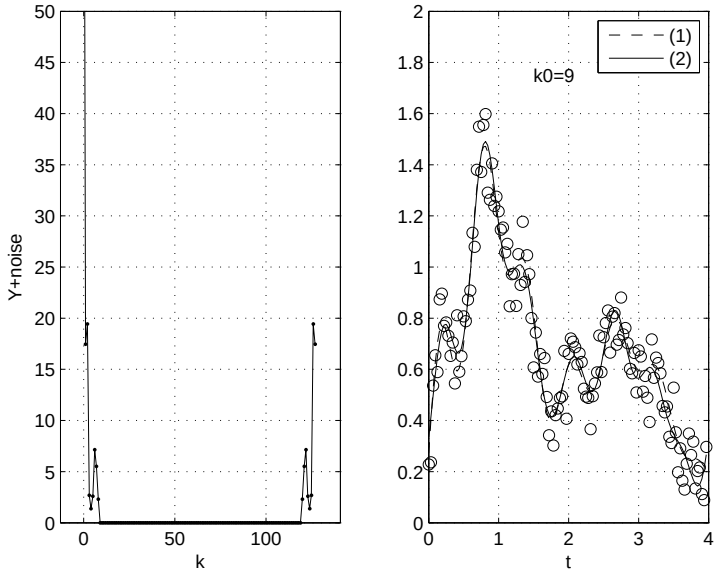



Abbildung 3.8: Demonstration einer Datenglättung mittels FT:
(a) Durchführung der Dirichlet-Dämpfung im Fourierraum für $k_0 = 9$,
(b) Wirkung dieser Dirichlet-Dämpfung im realen Zeitraum:
strichlierte Kurve: ungestörtes Signal;
volle Kurve: Dirichlet-behandeltes Signal.

### 3.4.3 Frequenz-Analyse

Ausgangspunkt für die folgenden Überlegungen ist die Interpolationskurve (3.15)

$$I(t) = \frac{1}{n}\sum_{k=0}^{n-1} Y_k \exp\left[-i\frac{2\pi kt}{P}\right] = \frac{1}{n}\sum_{k=0}^{n-1} Y_k \exp\left(-i2\pi\nu_k t\right) .$$

Die Frequenzen, die in die obige Entwicklung einbezogen werden, lauten

$$\nu_k = \frac{k}{P} \qquad \text{mit} \quad k = 0, \ldots, n-1$$

mit $n$ als Zahl der Meßpunkte und $P$ als Periode des Meß-Signals.

Aus dieser Darstellung geht auch hervor, warum man diese Fourier-Transformation als Transformation *aus dem Zeitraum in den Frequenzraum* bezeichnet. Mathematisch völlig äquivalent ist auch die Transformation *vom Ortsraum in den Wellenzahlraum.*

Nun soll angenommen werden, daß nicht nur die gegebenen (gemessenen) Werte $y_j$, sondern die gesamte Funktion $f(t)$, welche 'hinter diesen Werten steht', reell ist. In diesen Fall interessiert vor allem der Realteil der Interpolationsfunktion, nämlich

$$\Re\{I(t)\} = \sum_{k=0}^{n-1}\left[\Re\left\{\frac{Y_k}{n}\right\}\cos\left(\frac{2\pi kt}{P}\right) + \Im\left\{\frac{Y_k}{n}\right\}\sin\left(\frac{2\pi kt}{P}\right)\right] . \qquad (3.24)$$

- Man kann nun sagen: die Funktion $f(t)$ besteht aus *cosinus*-Anteilen mit den Amplituden 'Realteil von $Y_k/n$' und aus *sinus*-Anteilen mit den Amplituden 'Imaginärteil von $Y_k/n$'.

Es stellt sich nun die Frage, ob die Interpolation Glg. (3.24) die bestmögliche ist, d.h., ob diese Funktion die Meßwerte wirklich so glatt wie möglich verbindet (vgl. Abschnitt 3.1).
Die Antwort ist nein, und die Begründung dafür finden Sie in der Abb. 3.9. Wegen der Periodizität der Interpolationsfunktion sind auch die Fourierkoeffizienten $Y_k$ periodisch, und zwar mit der Periode $n$, d.h. es gilt:

$$Y_k = Y_{k+\mu n} \qquad (3.25)$$

für jede beliebige ganze Zahl $\mu$.

Man kann nun zeigen, daß *jede zusammenhängende Gruppe von n Fourierkoeffizienten mit den zugehörigen Frequenzen eine Interpolationsfunktion durch die gegebenen Punkte ergibt*, egal in welchem Bereich der Frequenzachse diese Gruppe liegt. Natürlich erhält man dabei jedesmal eine verschiedene Interpolationsfunktion!

Wenn man nun fragt, welche der möglichen Interpolationen diejenige sein wird, welche die geringste Neigung zur Oszillation zwischen den Stützpunkten hat, so ist die Antwort einfach: natürlich jene, welche die (dem Betrag

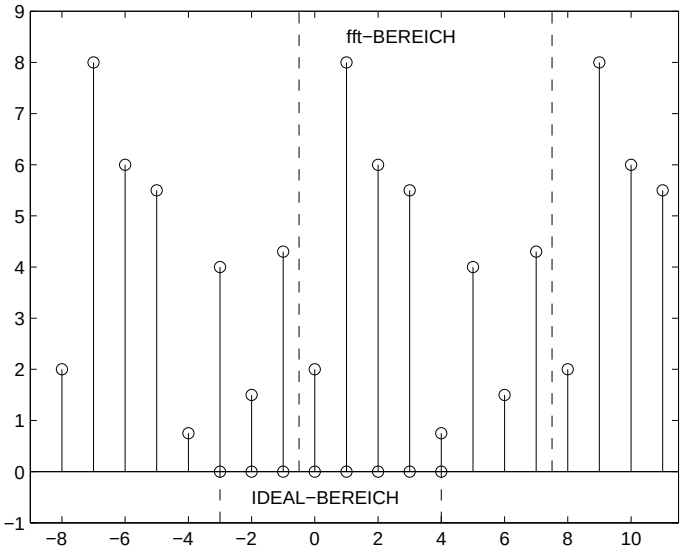



Abbildung 3.9: Periodizität der Fourierkoeffizienten im $k$=Raum für $n$=8. Es sind zwei Bereiche markiert: 'fft-BEREICH' bedeutet den Bereich der Fourierkoeffizienten, den die Fourier-Programme liefern ($k$=0...7), und 'IDEAL-BEREICH' bedeutet den Bereich $k$=–3,–2,–1,0,1,2,3,4, der die 'glattestmögliche' Interpolationsfunktion ergibt.

nach) kleinsten Frequenzen enthält. Wenn man also $n$ Koeffizienten benötigt, wird der ideale Frequenzbereich *um die Frequenz Null* liegen, also im Bereich $-n/2 < k \leq +n/2$.
Um diese ideale Interpolation formelmäßig darzustellen, bedarf es nur einer einfachen Umformung der Glg. (3.24):

$$\Re\{I(t)\} = \sum_{k=-(n/2)+1}^{-1} \left[ \Re\left\{\frac{Y_k}{n}\right\} \cos\left(\frac{2\pi kt}{P}\right) + \Im\left\{\frac{Y_k}{n}\right\} \sin\left(\frac{2\pi kt}{P}\right) \right] + \\ + \sum_{k=0}^{n/2} \left[ \Re\left\{\frac{Y_k}{n}\right\} \cos\left(\frac{2\pi kt}{P}\right) + \Im\left\{\frac{Y_k}{n}\right\} \sin\left(\frac{2\pi kt}{P}\right) \right] .$$

Führt man nun im ersten Term der obigen Gleichung die Index-Transformation

$$k' = n + k$$

durch, so ergibt sich weiter

$$\Re\{I(t)\} = \sum_{k'=(n/2)+1}^{n-1} \left[ \Re\left\{\frac{Y_{k'-n}}{n}\right\} \cos\left(\frac{2\pi (k'-n)t}{P}\right) + \Im\left\{\frac{Y_{k'-n}}{n}\right\} \sin\left(\frac{2\pi (k'-n)t}{P}\right) \right] + \\ + \sum_{k=0}^{n/2} \left[ \Re\left\{\frac{Y_k}{n}\right\} \cos\left(\frac{2\pi kt}{P}\right) + \Im\left\{\frac{Y_k}{n}\right\} \sin\left(\frac{2\pi kt}{P}\right) \right] .$$

Unter Berücksichtigung der Periodizität der $Y_k$ (3.25) gilt

$$Y_{k'-n} = Y_{k'} ,$$

und man erhält (mit $k \to k'$) das Ergebnis

$$\Re\{I(t)\}_{\text{ideal}} = \sum_{k=0}^{n-1} \left[ \Re\left\{\frac{Y_k}{n}\right\} \cos(2\pi\nu_k t) + \Im\left\{\frac{Y_k}{n}\right\} \sin(2\pi\nu_k t) \right] \tag{3.26}$$

mit

$$\nu_k = \frac{k}{P} \qquad \text{für} \quad k = 0, \ldots, \frac{n}{2} \tag{3.27}$$

und

$$\nu_k = \frac{k-n}{P} \qquad \text{für} \quad k = \frac{n}{2}+1, \ldots, n-1 \tag{3.28}$$

Diese Darstellung hat den Vorteil, daß in ihr nur die $Y_k$-Werte vorkommen, die von *four1.c, ftt.f90 und fft_sor.m* geliefert werden, und zwar genau in der entsprechenden Reihenfolge!

Das heißt natürlich nicht, daß (3.26)–(3.28) die (unbekannte) Funktion $f(t)$ 'hinter den Stützpunkten' ideal repräsentieren. Das geht schon daraus hervor, daß nur diskrete Frequenzen mit einer Auflösung von

$$\Delta\nu = \frac{1}{P} \tag{3.29}$$

sowie innerhalb des Frequenz-Intervalls

$$-\left(\frac{n}{2P}\right) < \nu \leq \left(\frac{n}{2P}\right) \tag{3.30}$$

erfaßt werden. Die 'Grenzfrequenz' $\nu_{\text{Ny}} = n/(2P)$ wird in der Literatur 'Nyquist-Frequenz' genannt.

Man kann also, was die Qualität einer Frequenzanalyse betrifft, wie folgt zusammenfassen:

- große Periode $P$ $\longrightarrow$ hohe Frequenz-Auflösung,
- große Punktanzahl $n$ $\longrightarrow$ großer Frequenzbereich.

**Aliasing effect**

Wenn das zu analysierende Signal eine Frequenz-Komponente $\nu_0 = k_0/P$ enthält, welche außerhalb des Bereichs (3.30) liegt, kommt es zu einem störenden Effekt, der darin besteht, daß die entsprechende Amplitude mit dem Index $k_0$ *in den Nyquist-Bereich hineingespiegelt wird*, d.h. es wird eine Frequenz-Komponente

$$\nu_0' = \frac{n-k_0}{P} \tag{3.31}$$

angezeigt, die tatsächlich nicht existiert. Dieses *aliasing effect* wird in Abb. 3.10 schematisch dargestellt.

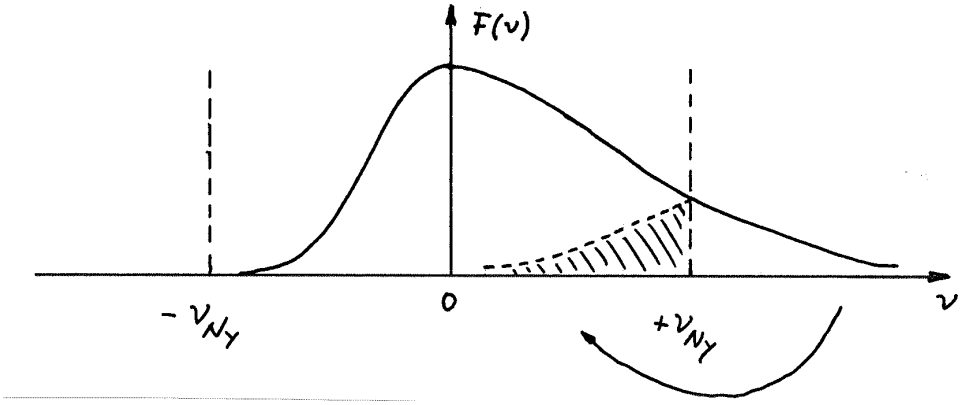



Abbildung 3.10: Der *aliasing effect* bei der Frequenzanalyse (schematisch).

**Ein einfaches Beispiel zur Frequenz-Analyse**

Im folgenden sollen die Zusammenhänge dieses Abschnittes an Hand eines Testbeispiels erläutert werden. Zu diesem Zweck definiert man eine analytische Funktion, die nur aus Cosinus- und Sinus-Termen besteht, und berechnet aus dieser Funktion eine Reihe von Werten, die man als Daten für eine Frequenz-Analyse mittels FT nimmt. Man erwartet, daß die FT-Analyse genau die Amplituden und Frequenzen der Testfunktion liefert.

Testfunktion:

$$f(t) = 2\cos(2\pi \cdot 2 \cdot t) - 3\sin(2\pi \cdot 4 \cdot t) - cos(2\pi \cdot 4 \cdot t) + 2\sin(2\pi \cdot 7 \cdot t)$$

Die Frequenzen sowie die Amplituden lauten also:

| Frequenz (Hz) | cos-Ampl. | sin-Ampl. |
|---|---|---|
| 2 | +2 | -- |
| 4 | -1 | -3 |
| 7 | -- | +2 |

Die Periode der Testfunktion ist $P$=1 s, und es wurden $n$=64 Stützpunkte berechnet. Für die Frequenz-Analyse bedeutet das eine Frequenzauflösung von $\Delta\nu$=1 Hz sowie einen Frequenzbereich von –31 bis +32 Hz [s. Glgen. (3.29) und (3.30)].

Wenn man dieses Problem mit Hilfe des Programms FOUR1 angeht (ohne die in diesem Abschnitt beschriebene Frequenz-Optimierung), erhält man das Ergebnis

| Frequenz (Hz) | RT(Y)/n (cos) | IT(Y)/n (sin) |
|---|---|---|
| 2.000 | 1.0000 | 0.0000 |
| 4.000 | -0.5000 | -1.5000 |
| 7.000 | -0.0000 | 1.0000 |
| 57.000 | 0.0000 | -1.0000 |
| 60.000 | -0.5000 | 1.5000 |
| 62.000 | 1.0000 | 0.0000 |

Die ersten drei Frequenzen sind korrekt, die drei letzten Ergebnisse liegen weit über den Testvorgaben. Dennoch ist die entsprechende Sin-Cos-Kurve eine *Interpolationskurve* [s. Abb. 3.11 (a)], allerdings kommt es zwischen den Stützpunkten zu starken Oszillationen.

Führt man jedoch die Frequenz-Optimierung im Sinne der Glgen. (3.26)–(3.28) durch, ergibt sich das Ergebnis

| Frequenz (Hz) | RT(Y)/n (cos) | IT(Y)/n (sin) |
|---|---|---|
| 2.000 | 1.0000 | 0.0000 |
| 4.000 | -0.5000 | -1.5000 |
| 7.000 | -0.0000 | 1.0000 |
| -7.000 | 0.0000 | -1.0000 |
| -4.000 | -0.5000 | 1.5000 |
| -2.000 | 1.0000 | 0.0000 |

Dieses Resultat entspricht exakt der Testvorgabe, und zwar sowohl in bezug auf die Frequenzen als auch in bezug auf die Amplituden. So werden z.B. zwei Cosinusse angezeigt, jeweils mit der Amplitude 1.000, und mit den Frequenzen +2 und –2. Das bedeutet:

$$\cos(2\pi \cdot 2 \cdot t) + \cos(-2\pi \cdot 2 \cdot t) = 2\cos(2\pi \cdot 2 \cdot t)\,,$$

d.h. der Kosinus-Term mit der Frequenz 2 Hz besitzt die Gesamtamplitude 2, was exakt mit dem ersten Term der Testfunktion übereinstimmt.
Ebenso werden z.B. zwei Sinusse angezeigt, einer mit Frequenz +4 Hz und Amplitude –1.5, und ein zweiter mit Frequenz –4 Hz und der Amplitude +1.5. Dies ergibt

$$-1.5\sin(2\pi \cdot 4 \cdot t) + 1.5\sin(-2\pi \cdot 4 \cdot t) = -3\sin(2\pi \cdot 4 \cdot t)\,,$$

also exakt den zweiten Term der Testfunktion, usw.
Die entsprechende Interpolationsfunktion [s. Abb. 3.11 (b)] entspricht genau der Testkurve.

Zum Abschluß soll noch der *aliasing effect* demonstriert werden. Angenommen, die Testfunktion laute

$$f(t) = 2\cos(2\pi \cdot 2 \cdot t) - 3\sin(2\pi \cdot 4 \cdot t) - cos(2\pi \cdot 4 \cdot t) + 2\sin(2\pi \cdot 55 \cdot t)\,.$$

D. h., sie ist bis auf den vierten Term identisch, der eine Frequenz von 55 Hz hat. Wegen der Periode von 1 s und den $n$=64 Stützpunkten hat die Nyquist-Frequenz den Wert 32 Hz. Die Frequenz des vierten Terms liegt also *außerhalb* des Nyquist-Bereiches.
Das FT-Ergebnis inklusive *Frequenz-Optimierung* lautet in diesem Fall

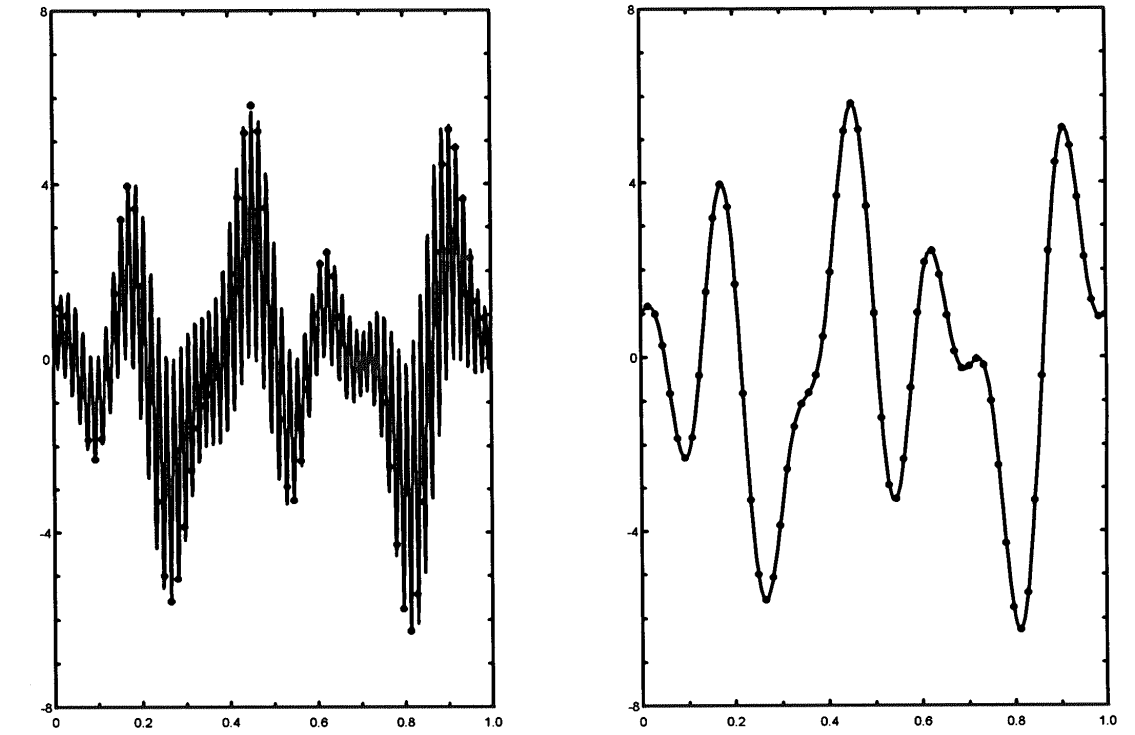



Abbildung 3.11: (a) FT-Frequenz-Analyse ohne Frequenz-Optimierung, (b) FT-Frequenz-Analyse mit Frequenz-Optimierung. Die Kreise sind die Stützpunkte.

| Frequenz (Hz) | RT(Y)/n (cos) | IT(Y)/n (sin) |
|---|---|---|
| 2.000 | 1.0000 | 0.0000 |
| 4.000 | -0.5000 | -1.5000 |
| 9.000 | 0.0000 | -1.0000 |
| -9.000 | -0.0000 | 1.0000 |
| -4.000 | -0.5000 | 1.5000 |
| -2.000 | 1.0000 | 0.0000 |

Wie Sie sehen, werden die ersten drei Terme der Testfunktion exakt erhalten, der letzte Term mit seinen 55 Hz scheint hingegen nicht auf. Statt dessen wird eine Frequenz von $\pm$ 9 Hz angezeigt, die in der Testfunktion nicht vorkommt. Ein typischer *aliasing effect* s. Glg. (3.31)] wegen

$$64 - 55 = 9\,.$$

Die Gesamtamplitude dieses *aliasing*-Terms ist, wie aus der obigen Tabelle hervorgeht, $-1 - (1) = -2$. In der entsprechenden Testfunktion hat dieser Term (bei 55 Hz) die Amplitude +2. Die Ursache dafür ist, daß beim *aliasing* eines Cosinus-Terms dieser unter Beibehaltung seines Vorzeichens in den Nyquist-Raum gespiegelt wird, während ein Sinus-Term dabei sein Vorzeichen ändert.

### 3.4.4 Weitere Stichworte zum Thema: Diskrete Fourier-Transformation

Wie auch in anderen Kapiteln dieses Skriptums muß ich mich auch bei der FT auf die wichtigsten Grundlagen beschränken. Auch hier möchte ich versuchen, Ihnen weitere – insbesondere für die Anwendung – wichtige Themen in meiner LV im SS *Ausgewählte Kapitel aus 'Numerische Methoden in der Physik'* zu vermitteln:

- Mehr zum Stichwort 'Faltung und Korrelation' (inkl. Anwendungsbeispiele).
- Fortgeschrittene Filtermethoden zur FT-Behandlung von statistisch fehlerhaften Daten.
- Grundlagen der Computer-Tomografie.

### 3.4.5 Software-Angebot

Da die Fourierentwicklung eine außerordentlich wichtige Methode darstellt, mathematische und physikalische Probleme zu behandeln, ist das Angebot an einschlägigen Programmen besonders groß. Ich kann daher im folgenden nur sehr allgemeine Informationen geben:

**FORTRAN-User** :

- F-Programme für FFT in den 'Numerical Recipes' [9] bzw. im Buch von Engeln-Müllges [6]. Zugang auf dem PCs des Computerraums Physik:

  `/usr/local/numeric/numrec/f` bzw.
  `/usr/local/numeric/num_alg/f`

- Schauen sich sich im NETLIB das Programmpacket *FFTPACK* an.

**C-User** :

- Eine Reihe von C-Programmen zum Thema *Fast-Fourier-Transform* finden Sie in den 'Numerical Recipes C' [10] sowie in [6] bzw. auf den 'Physik-PCs' unter

  `/usr/local/numeric/numrec/c` bzw.
  `/usr/local/numeric/num_alg/c`

- Ein interessantes C-Programm fuer FFT können Sie aus dem Internet herunterladen, nämlich das C-Programm *fftw.c*. Die Autoren M. Frigo und S. G. Johnson behaupten, daß es schneller sei als alle anderen Programme am Markt. Außerdem ist es nicht an die Bedingung $n = 2^m$ gebunden.
  Adresse: www.fftw.org

**Mathematica:** Fourier$[a_0, a_1, \ldots]$ InverseFourier$[b_0, b_1, \ldots]$
für alle $n$, wobei aber *the efficiency significantly increases when n is a power of two.*

Ein Beispiel: *Smoothing of noised data by convolution with a kernel:*[6]

```
data=Table[N[Bessel][1,10 n/256] + 0.2 (Random[]-1/2)],{n,256}];
kern=Table[N[Exp[-200 (n/256)^2]],{n,256}];
conv=InverseFourier[Fourier[data]/Fourier[kern]];
ListPlot[Chop[conv]]
```

**MATLAB:** Das Programm *fft* habe ich bereits vorgestellt. Dazu gibt es noch ein entsprechendes Programm zur inversen FFT mit Namen *ifft*.

Vergessen Sie bitte nicht meinen Rat, anstelle dieser Routinen die von mir variierten Programme *fft_sor* bzw. *ifft_sor* zu verwenden.

Weitere Fourier-Transform-Programme von MATLAB:

| | |
|---|---|
| fft2 | zweidim. Fast-Fourier-Transform |
| ifft2 | inverse zweidimensionale FFT |
| fftshift | Programm zur Frequenzverschiebung (vgl. Abschnitt 3.4.3). |

[6] S. Wolfram, *Mathematica*, Addison-Wesley Publishing Comp., 1991, S. 681ff.