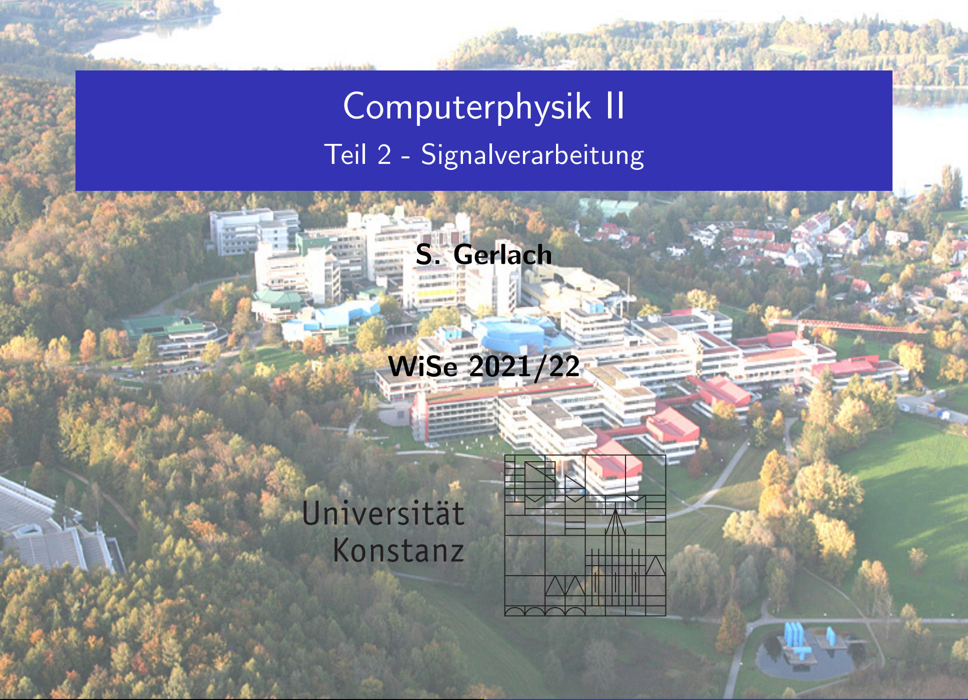


Computerphysik II

Teil 2 - Signalverarbeitung

S. Gerlach

WiSe 2021/22

Universität Konstanz

INHALT

1. Entwicklung von Funktionen
2. DFT und Anwendungen
3. Signalverarbeitung und -analyse
4. Bildbearbeitung
5. Bildanalyse

Entwicklung nach Basisfunktionen

Viele Signale sind periodisch und glatt.
Idee: Entwicklung nach einfachen, bekannten Basisfunktionen.
Basis: Sinus/Cosinus bzw. kompl. Exponentialfkt.
$\rightarrow$ Reelle **Fourier-Reihe**:

$$f_n(x) = \frac{a_0}{2} + \sum_{j=1}^{n} (a_j \cos(k_j x) + b_j \sin(k_j x)),\ k_j = \frac{2\pi}{L} j$$

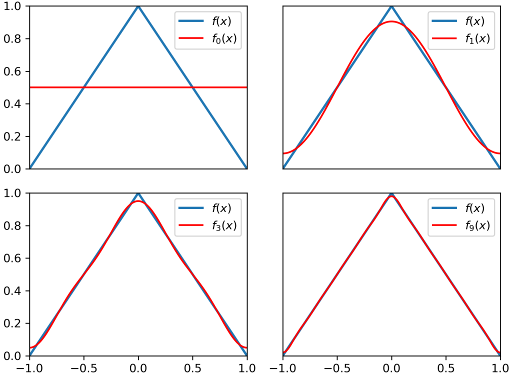
mit

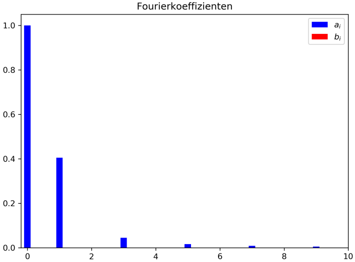
$$\begin{aligned} a_j(x) &= \frac{2}{L} \int_a^b f(x) \cos(k_j x)\, \mathrm{d}x \quad (j \geq 0), \\ b_j(x) &= \frac{2}{L} \int_a^b f(x) \sin(k_j x)\, \mathrm{d}x \quad (j > 0), \end{aligned} \tag{1}$$

Komplexe Fourier-Reihe:

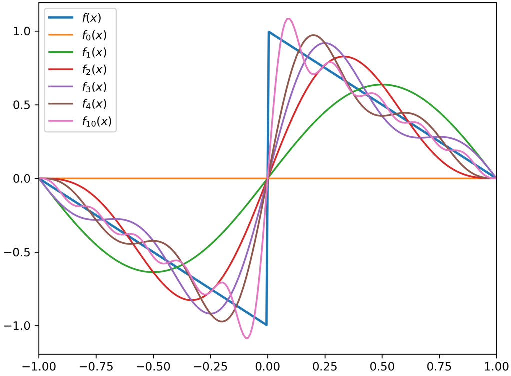
$$f_n(x) = \sum_{j=-n}^{n} c_j \mathrm{e}^{\mathrm{i} k_j x}, c_j = \frac{1}{L} \int_a^b f(x) \mathrm{e}^{-\mathrm{i} k_j x}\, \mathrm{d}x.$$

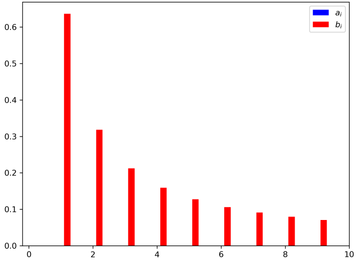
Fourier-Reihe - Beispiel 1



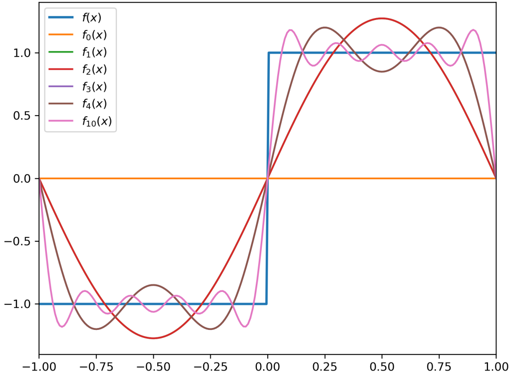


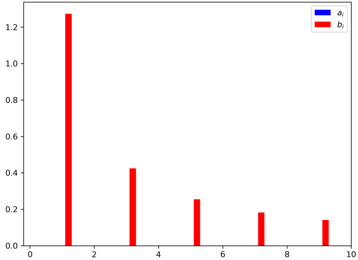
Fourier-Reihe - Beispiel 2



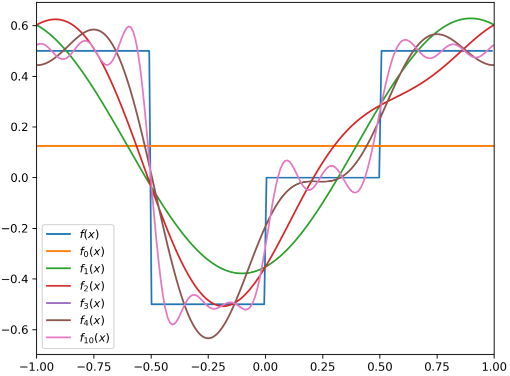


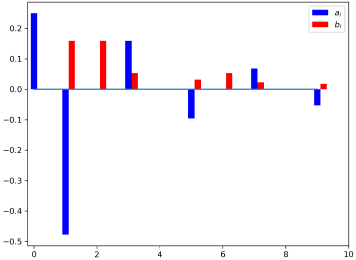
Fourier-Reihe - Beispiel 3





Fourier-Reihe - Beispiel 4

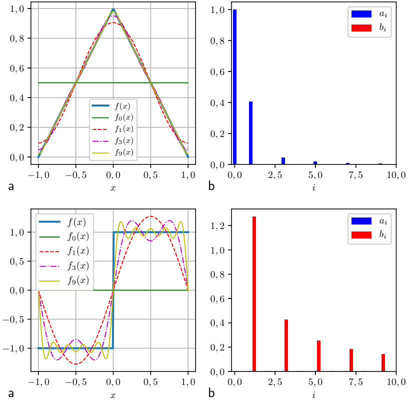




Fourier-Reihe

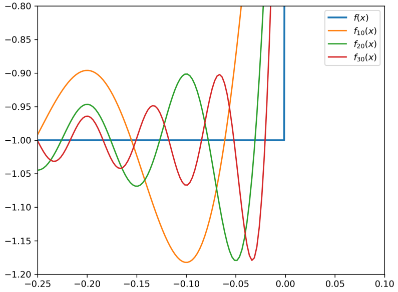
Je höher die Ordnung (n), desto höher die Frequenz der Basisfunktionen
$\rightarrow$ Schnelle Konvergenz erwünscht.
Man kann zeigen:

- Symmetrie/Antisymmetrie: $b_i = 0 / a_i = 0$
- Kovergenz der Fourierkoeff. **quadratisch wenn stetig**, sonst linear.



Fourier-Reihe

Zusätzliches Problem mit Sprungstellen: **Gibbs-Phänomen**: Überschwingen bei Sprungstellen



Zerlegung in Fourierkoeff. eindeutig. $\rightarrow$ Transformation **Ort in Impuls** (k) bzw. **Zeit in Frequenz**.

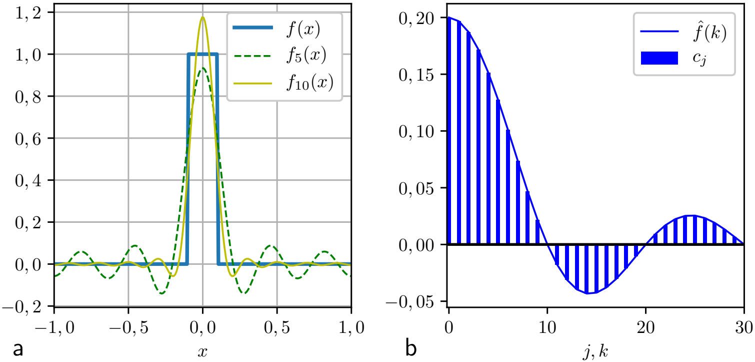
* **Übung**: Entwicklung nach Legendre-Polynomen (s. Buch, Kap. 9.4)

Fouriertrafo

Verallgemeinerung auf nicht-period. Funktionen: $L/T \to \infty$
(komplexe) Fourierkoeff.:

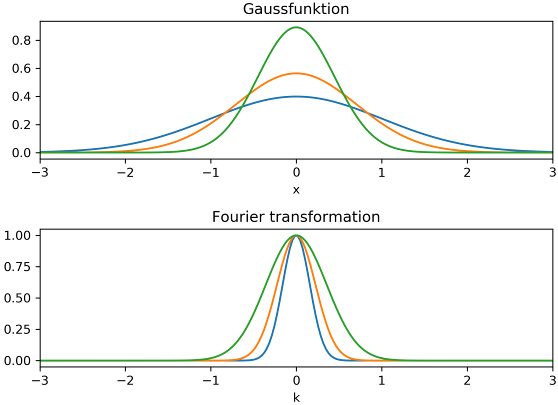
$$c_j = \frac{1}{L}\int_{-L/2}^{L/2} f(x)\mathrm{e}^{-\mathrm{i}k_j x}\,\mathrm{d}x \overset{L\to\infty}{\to} \int_{-\infty}^{\infty} f(x)\mathrm{e}^{-\mathrm{i}kx}\,\mathrm{d}x = \hat{f}(k)$$

(kont.) **Fouriertransformation**



Fouriertrafo

Analytische Berechnung mit SymPy:
`sympy.fourier_transform(f(x),x,k)`



→ mehr Beispiele: Laplace, sinc, Wellenpaket, ...

Diskrete Fouriertrafo

Im Computer nur diskrete Werte möglich, d. h. **Diskretisierung** nötig:
$x = x_0 + k\Delta x \rightarrow$ (Herleitung im Buch)
Diskrete Fouriertrafo (DFT):

$$\hat{f}_j = \sum_{k=0}^{N-1} f_k \mathrm{e}^{-\mathrm{i}2\pi\frac{jk}{N}} \quad (j = 0, \cdots, N-1)$$

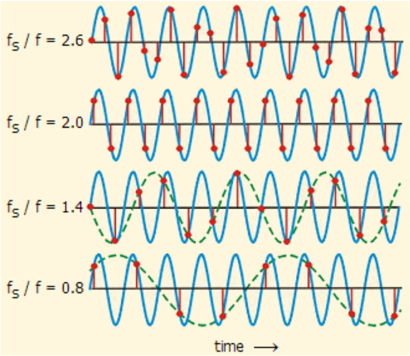
$$f_k = \sum_{j=0}^{N-1} \hat{f}_j \mathrm{e}^{\mathrm{i}2\pi\frac{jk}{N}} \quad (k = 0, \cdots, N-1)$$

DFT-Berechnung: **FFT-Algorithmen**. (Fast-Fourier-Trafo)
`numpy.fft.fft()`, `numpy.fft.ifft()`,
`scipy.fftpack.fft()`, `scipy.fftpack.ifft()`
Reelle DFT: N (reelle) Werte $\rightarrow N/2 + 1$ komplexe Ergebnisse

DFT verstehen

Beispiel: **Frequenzanalyse**
ΔT - Abstand der Messungen (Samples)
$\rightarrow$ **Samplefrequenz**: $\nu = 1/\Delta T$

Maximal messbare Frequenz (3 Messpunkte): $\nu_{\mathrm{Ny}} = \frac{\nu}{2} = \frac{1}{2\Delta T}$
(**Nyquist-Frequenz**)



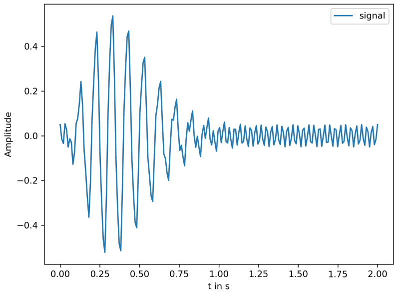
DFT verstehen

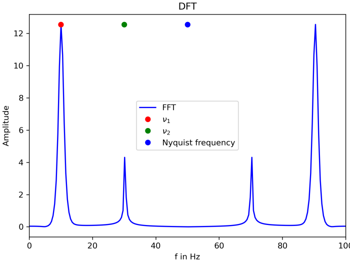
Beispiel: **Frequenzanalyse**
ΔT - Abstand der Messungen (Samples)
$\rightarrow$ **Samplefrequenz**: $\nu = 1/\Delta T$

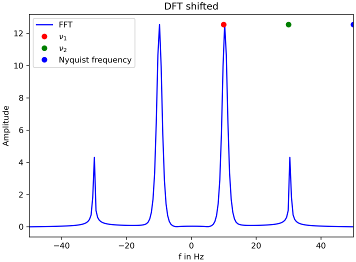
$\rightarrow$ Je höher die Samplefrequenz, desto größer die messbaren **Frequenzen**.

Frequenzabstand: $\Delta\nu = \frac{\nu}{N} = \frac{1}{N\Delta T} = \frac{1}{T}$
$\rightarrow$ Je länger die Messzeit, desto genauer die **Frequenzauflösung**.

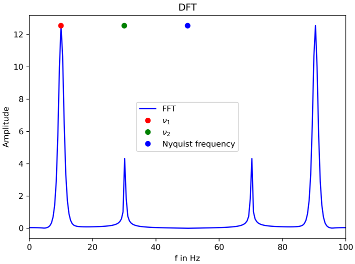
DFT verstehen

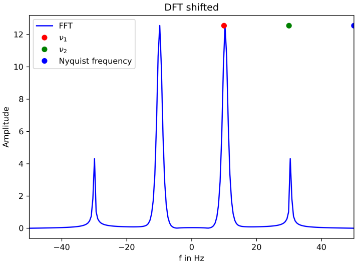


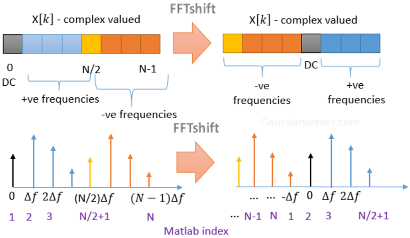




DFT verstehen

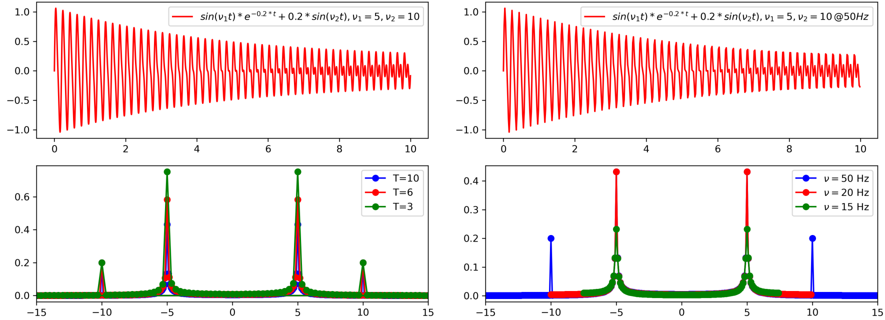






`numpy.fft.fftshift(), numpy.fft.fftfreq()`

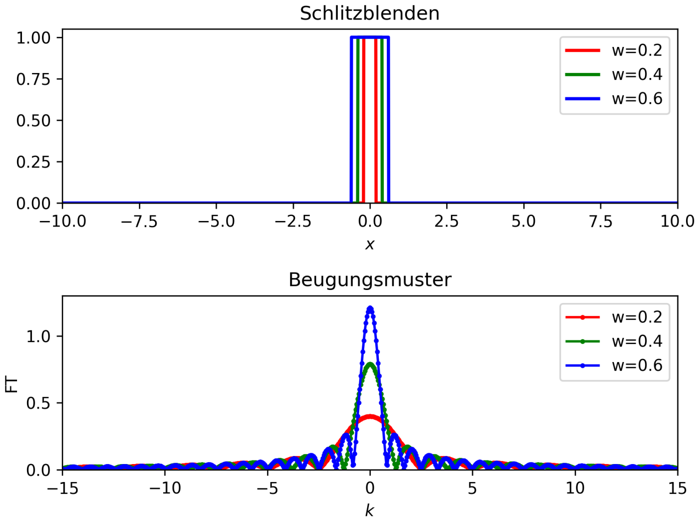
Variation: Messzeit, Samplefrequenz




```
dt = T/N
x = fftfreq(N, dt)
Y = fft(y) * 2.0/N
pl.plot(x,abs(Y))
```

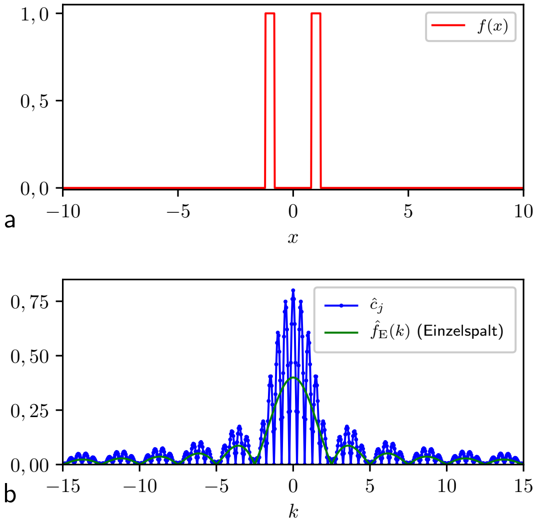
Anwendung: Beugung

Beispiel: **Beugungsmuster eines Spaltes**:



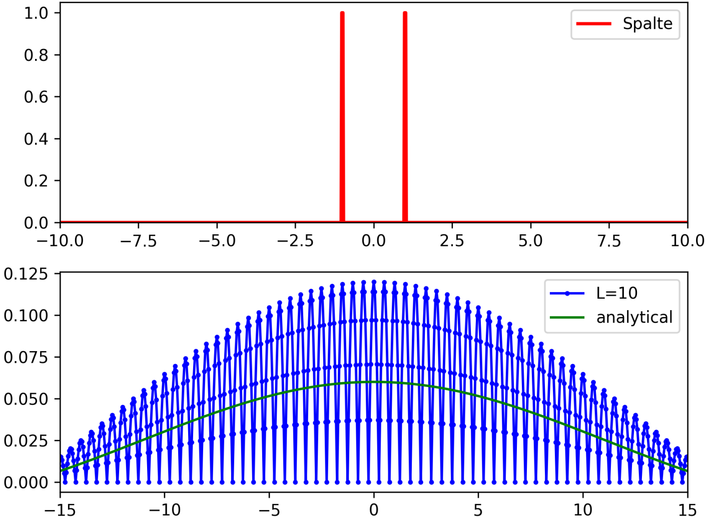
Anwendung: Beugung

Beispiel: **Beugungsmuster eines Doppelspaltes**:



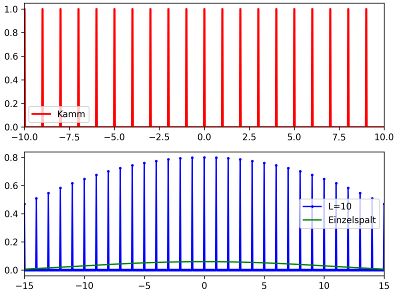
Anwendung: Beugung

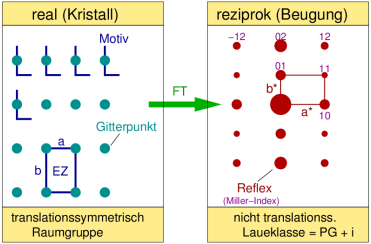
Interferenzmuster (sehr schmale Spalte):



Anwendung: Beugung

Gitter (Kamm): $\rightarrow$ Reziprokes Gitter

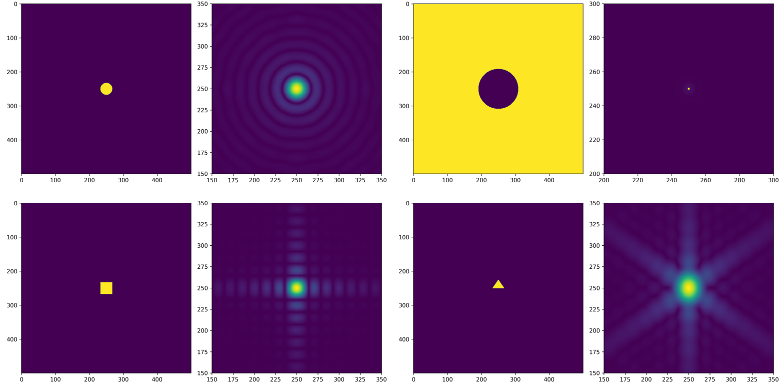




2D-DFT:

$$\boxed{\hat{a}_{kl} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} a_{mn} \mathrm{e}^{-\mathrm{i}2\pi\left(\frac{mk}{M}+\frac{nl}{N}\right)}}$$

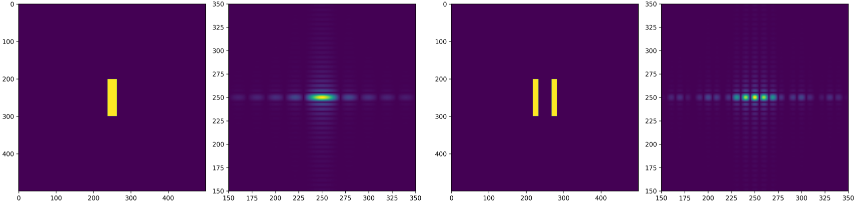
Lochblende, Poisson-Fleck, Rechteck-, Dreieckblende:



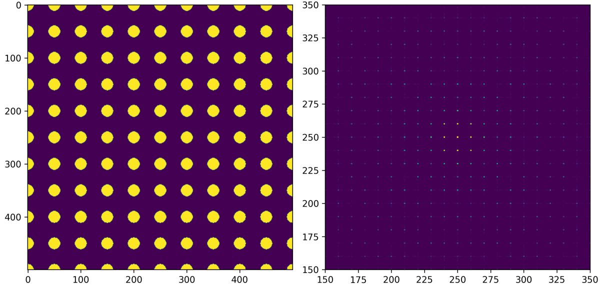
Sharp edges are represented by streaks in the FT perpendicular to the edge.

Anwendung: 2D Beugung

Realer Spalt/Doppelspalt:



Kreis-Gitter (große - kleine Strukturen invertiert!), `log(|FT| + X)`:



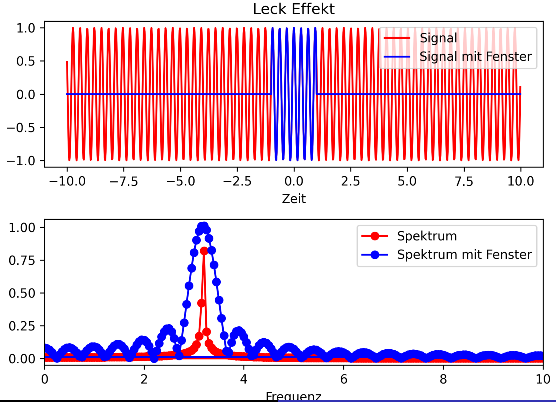
Übersicht DFT berechnen

Python:
`fft()/ifft()`, `fft2()/ifft2()`, `fftn()/ifftn()`
`fftshift()/ifftshift()`, `fftfreq()`
Optimiert: `numpy.fft`, `scipy.fftpack`, `pyfftw`

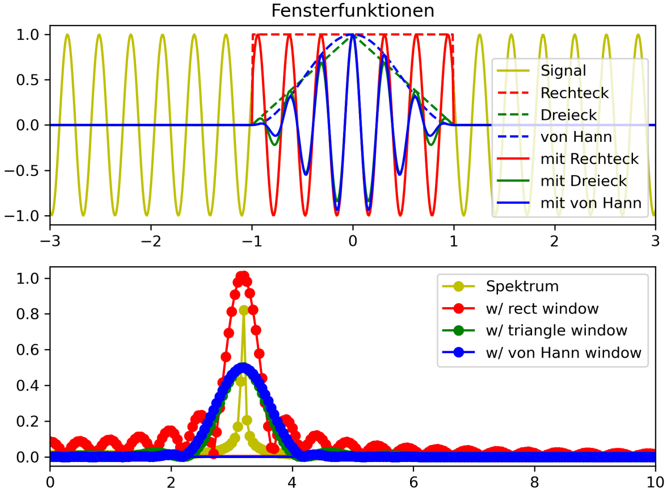
C:
FFTW - Fastest Fourier Trafo in the West
MKL - Math Kernel Library von Intel
@ GPU: cufft (CUDA)

Signalverarbeitung und -analyse: Wichtige Effekte

Realität: Signale (Messungen) immer endlich und damit z.B. im Frequenzbereich über einen weiten Bereich verteilt. entspricht Rechteckfilter in der Zeit, der zu einem schmalen Hauptmaximum, aber auch vielen Nebenmaxima im Frequenzbereich führt.
$\rightarrow$ **Leck-Effekt** (*spectral leakage*)

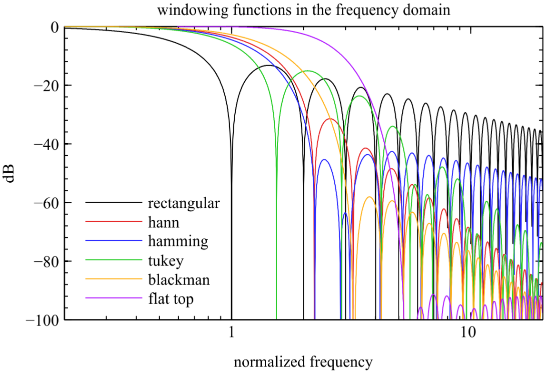


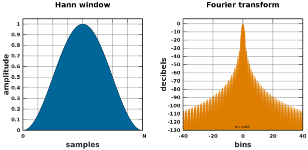
Lösung: Verwendung von sog. Fensterfiltern.
Beispiele: Dreieckfenster, von-Hann-Fenster, ... (WP)

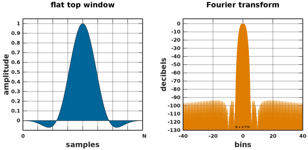


Signalverarbeitung und -analyse: Leck-Effekt

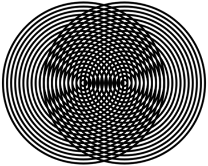
Vergleich: Breite/Nebenmaxima/Abklingen

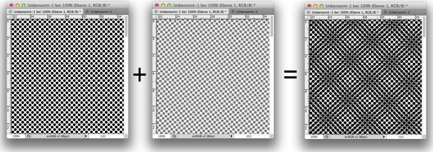


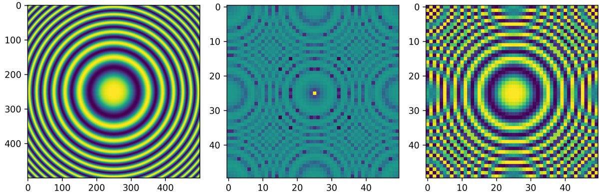




Alias-Effekt (Stroboskop-Effekt): Endliche Samplerate $\rightarrow$ Abschneiden hoher Frequenzen (**Abtasttheorem**, vgl. Nyquist-Frequenz)
$\rightarrow$ **Interferenz zw. Signal und Sample-/Abtastfrequenz**
Bei Resonanz: Stroboskop-Effekt, Moire-Muster, etc. (Auch bekannt aus Filmen: *Wagon-Wheel-Effekt*)

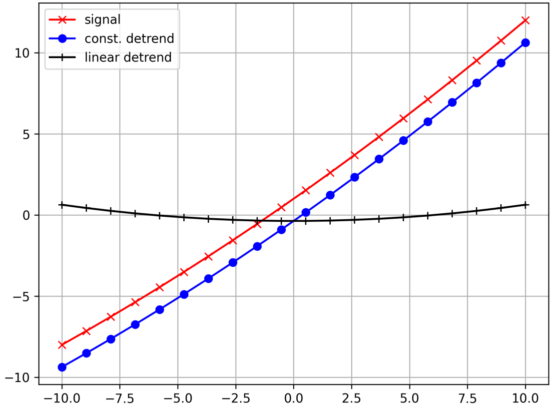






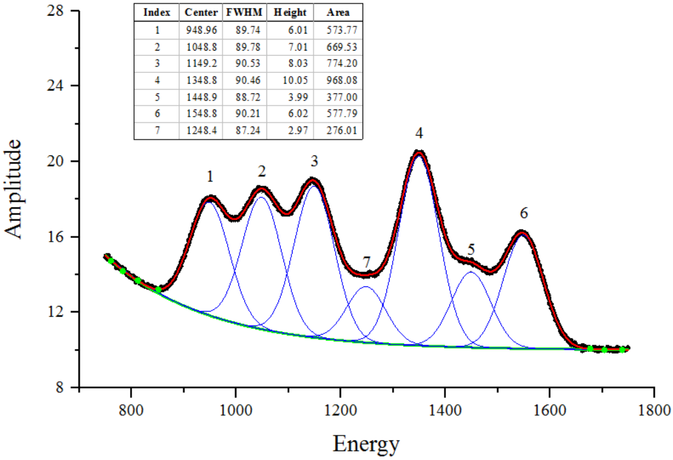
Signalverarbeitung und -analyse: Einfache Methoden

"Detrend": Entferne konstanten Untergrund bzw. Trend in den Daten. Dafür einfach lineare Anpassung anziehen: scipy.signal.detrend(data, type='constant'/'linear')



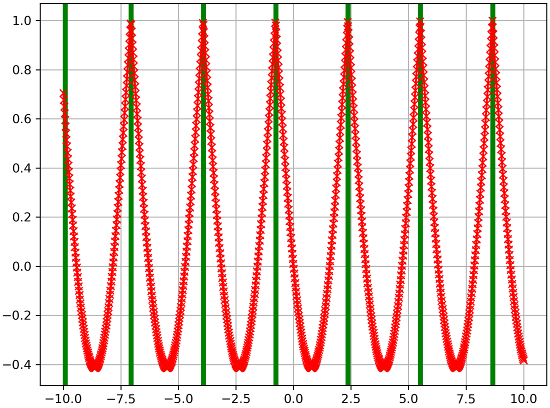
Signalverarbeitung und -analyse: Einfache Methoden

"Peak Find": Viele Signale enthalten Resonanzen (Spektren, etc.). Oft gesucht: Linienposition, -breite und -höhe
Idee: jeweils Anpassung mit Linienform ...



Index	Center	FWHM	Height	Area
1	948.96	89.74	6.01	573.77
2	1048.8	89.78	7.01	669.53
3	1149.2	90.53	8.03	774.20
4	1348.8	90.46	10.05	968.08
5	1448.9	88.72	3.99	377.00
6	1548.8	90.21	6.02	577.79
7	1248.4	87.24	2.97	276.01

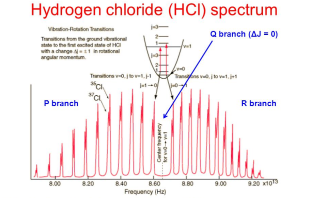
... oder spezielle Transformation (CWT):
`scipy.signal.find_peaks_cwt()`



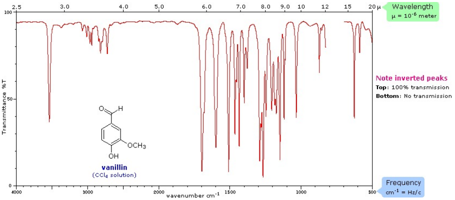
Signalverarbeitung und -analyse: Spektren

Mehr Beispiele:

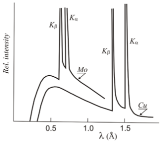
Molekülspektren

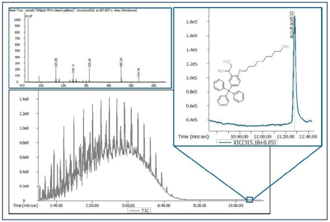


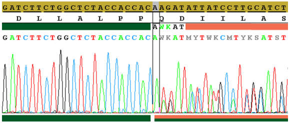
• Vibrational-rotational absorption spectrum of HCl: shows affect of two chlorine isotopes with slightly different mass



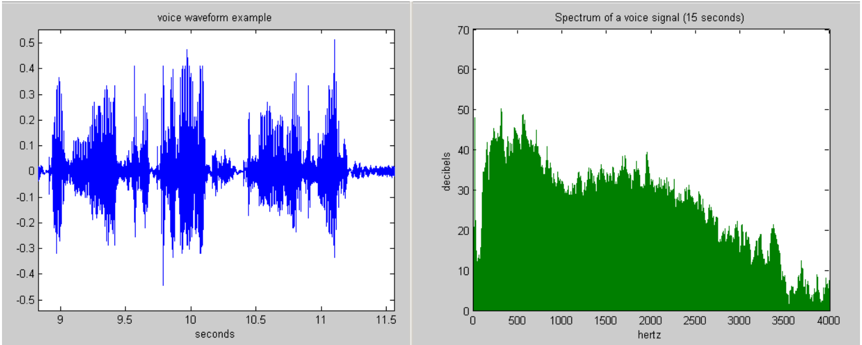
XRD, Massenspektrometer, Sequenzanalyse







Periodogramm: gleitende DFT eines Zeitsignals (mit Fenster) $\rightarrow$ power-spectral-density ($= \log |Amplitude|^2$)
`scipy.signal.periodogram()`



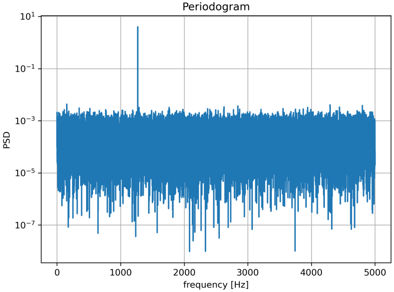
Optionen: `detrend`, `window`

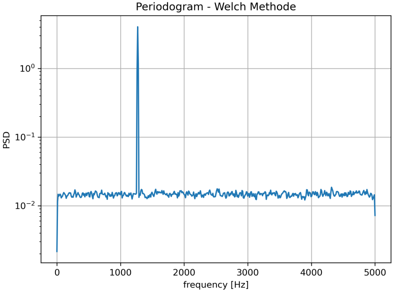
Signalverarbeitung und -analyse: Beispielmethoden

Optimiertes Periodigram: **Welch-Methode**:

`scipy.signal.welch()`

Überlappende Fensterfunktionen und Mittelung

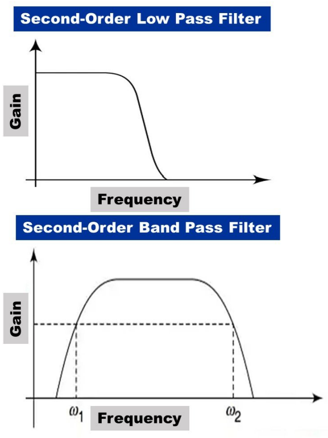




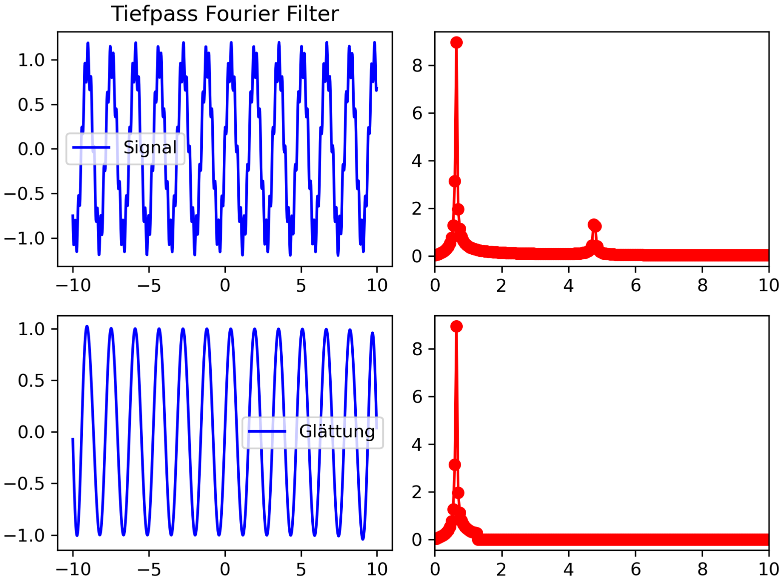
Optionen: Segmentgröße, Overlap, Fenster (default: Hann-Fenster)

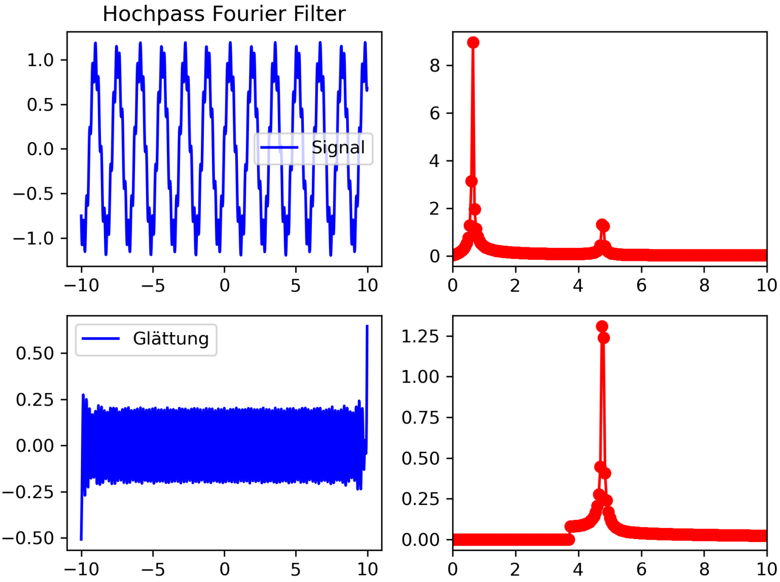
Signalverarbeitung und -analyse: Fourier-Filter

Idee: Verwende **Filter im Fourier-Raum** (Frequenzbereich)
Anwendungen: Hochpass-, Tiefpass-, Bandpass-, Bandblock-Filter

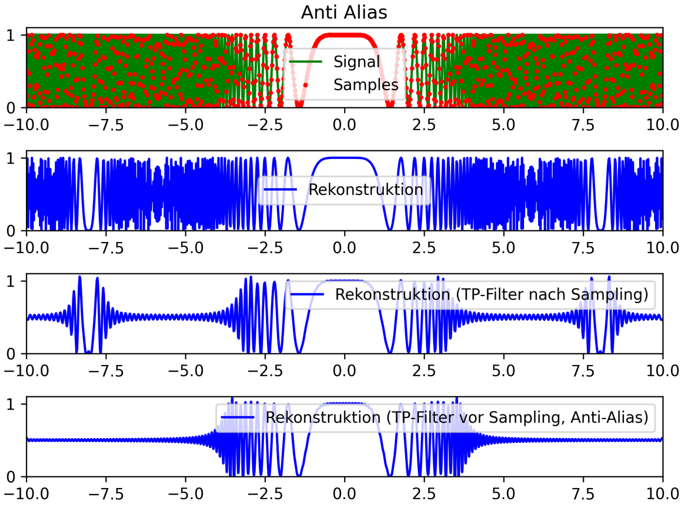


Signalverarbeitung und -analyse: Fourier-Filter

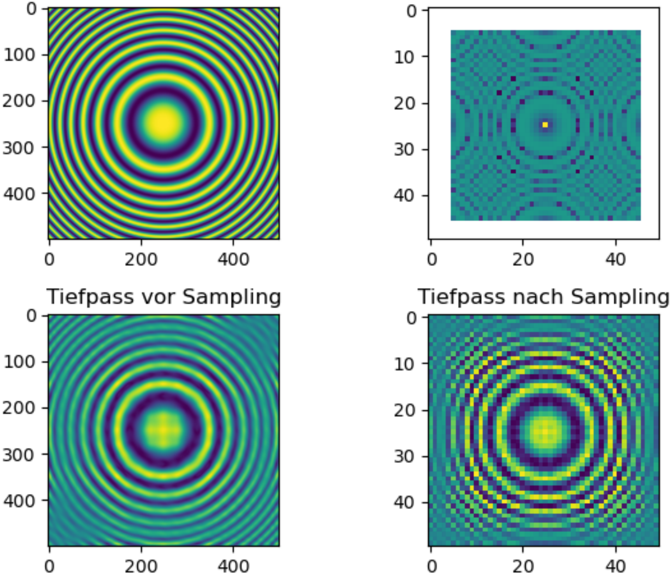



Hochpass Fourier Filter
Signal
Glättung
1.0
0.5
0.0
−0.5
−1.0
−10
−5
0
5
10
8
6
4
2
0
0.50
0.25
0.00
−0.25
−0.50
1.25
1.00
0.75

Anti-Alias: Aliaseffekt durch Filter verringern

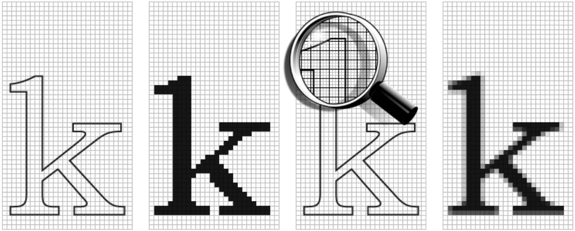


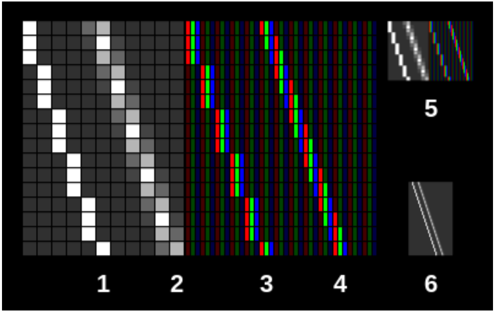
Signalverarbeitung und -analyse: Anti-Alias



Signalverarbeitung und -analyse: Anti-Alias

Auch bekannt bei Schriften (Raster) oder Monitoren (Pixel)

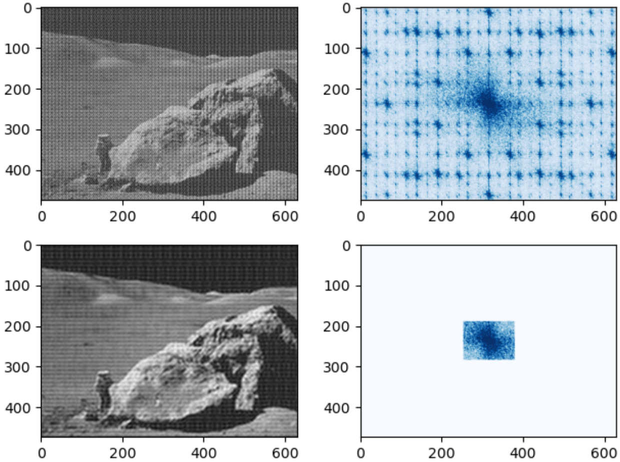




Viele Methoden, die Fourierfilter verwenden:

Beispiel 1: **Streifenfilter**

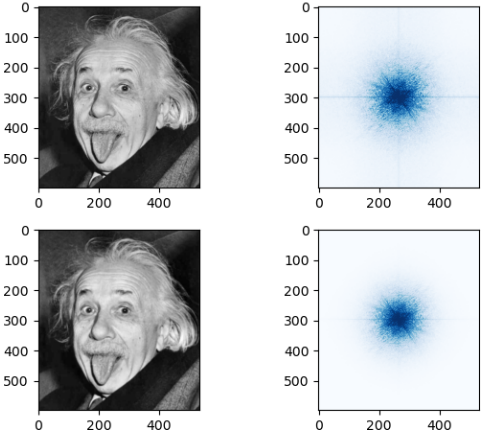
Entferne die Anteile im Fourierbild, die zu den Streifen führen



Beispiel 2: **Weichzeichnen**

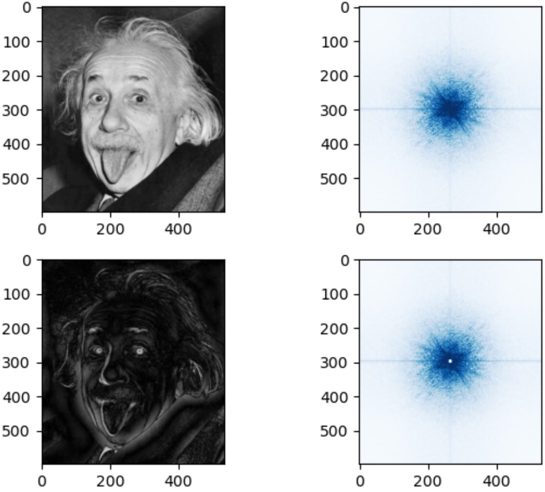
Entferne hochfrequente Anteile (Kanten, kleine Strukturen, etc.)

Um Aliaseffekte zu vermeiden ("Ringing"): verwende "weiche" Filter, z.B. Gauss



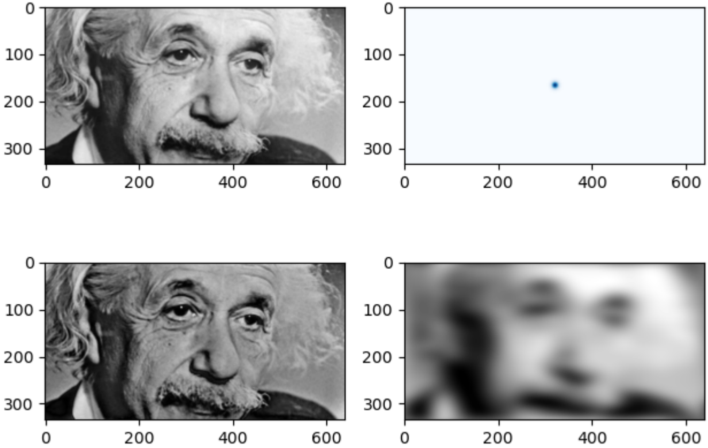
Beispiel 3: **Kantenfilter**

Verwende Hochpassfilter um Umrisse (Kanten) zu sehen



Beispiel 4: **Schärfen**

Idee: Weichzeichen des Bildes und Abziehen vom Originalbild



EVALUATION:

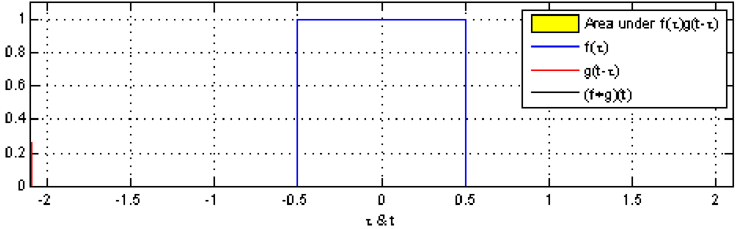
`https://evasys.uni-konstanz.de/evasys/online.php?pswd=Q8S6F`

Signalverarbeitung und -analyse: Faltung und Anwendungen

Faltung (*convolution*):

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,\mathrm{d}\tau.$$

Anschaulich: Eine (gespiegelte) Gewichtsfunktion ("Kernel") g wird über die Funktion f geschoben.



Diskrete Berechnung ($\mathcal{O}(N^2)$):

$$(f * g)_n = \sum_k^N f_k g_{n-k}$$

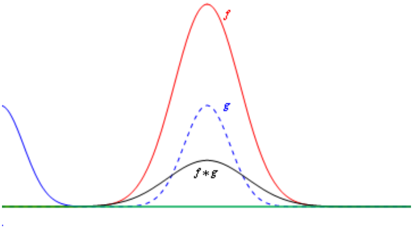
Besser: nutze **Faltungstheorem**

$$\boxed{f * g = \mathcal{F}^{-1}(k\mathcal{F}\{f\} \cdot \mathcal{F}\{g\})}$$

FFT mit $\mathcal{O}(N \log N)$ deutlich schneller als Summe (s.o.).

Ist g viel kürzer als f (g ist ein FIR - *Finite Impulse Response*):
Overlap-save oder *Overlap-add* Methode.
Beispiel: Gleitender Mittelwert durch Rechteck-FIR (*boxcar*)

Faltung einer Gaussfunktion mit Gauss-Kernel:

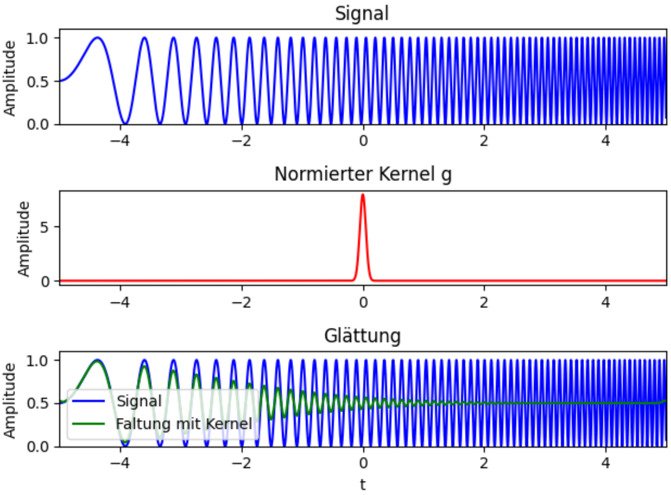


Typische Anwendung: Glättung durch Faltung mit Gauss-Kernel

Signalverarbeitung und -analyse: Glättung

Out-of-Focus Fotografie (**Bokeh**)
Unschärfe des Hintergrunds durch Linsenfunktion:

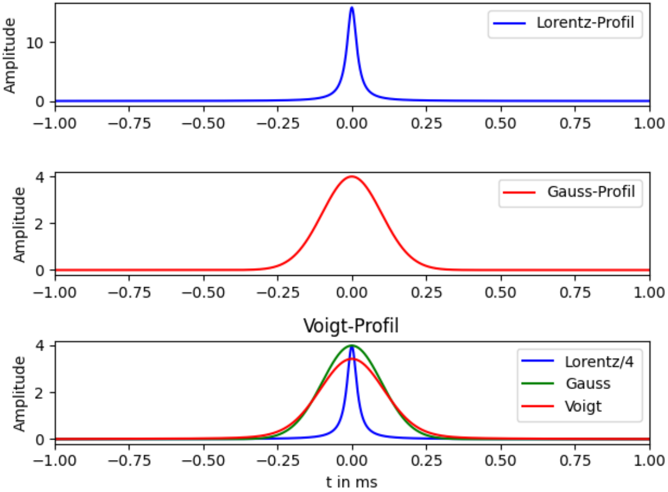
Signalverarbeitung und -analyse: Faltung Beispiel 1



`scipy.signal.convolve()`, `scipy.signal.fftconvolve()`

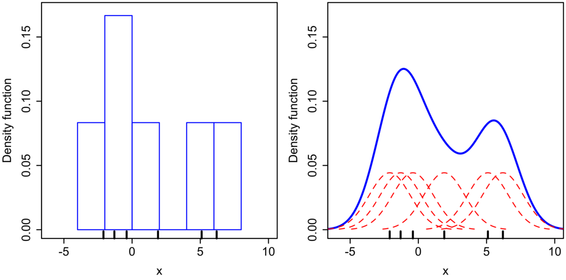
Signalverarbeitung und -analyse: Faltung Beispiel 2

Voigt-Profil: Faltung von Dopplerverbreiterung (Gauss) mit natürlicher Linienbreite (Lorentz)



Signalverarbeitung und -analyse: Faltung Beispiel 3

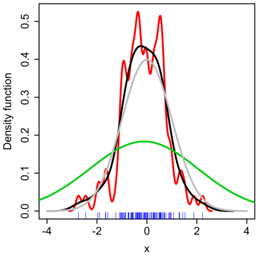
Kernel Density Estimation:



N-Datenpunkte KDE:

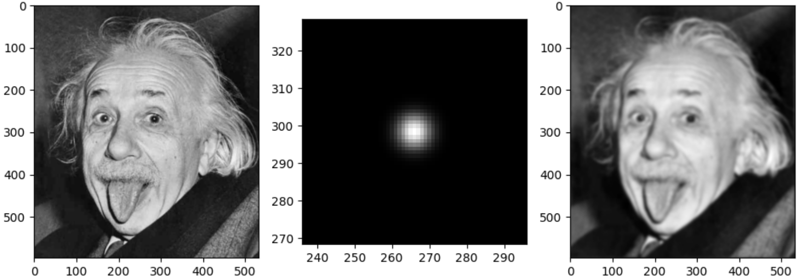
$$f_h(x) = \sum_{i}^{N} K_h(x - x_i)$$

Bestimmung **Bandbreite** h: *Rule of Thumb* oder Berechnung durch Fehlerminimierung.



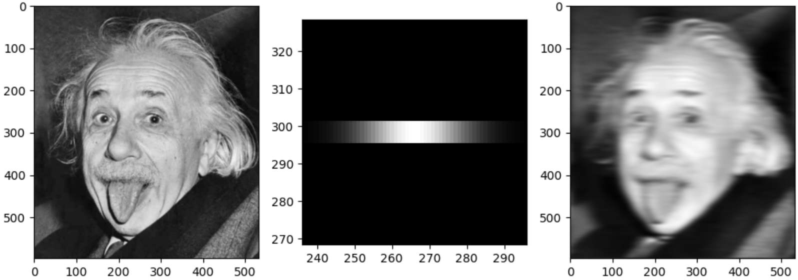
Signalverarbeitung und -analyse: Faltung und Anwendung

2D: Kernel sog. *point spread function* (**PSF**), da jeder Punkt/Pixel mit dem Kernel multipliziert wird.
2D-Gauss-Kernel: **Gauss-Glättung**

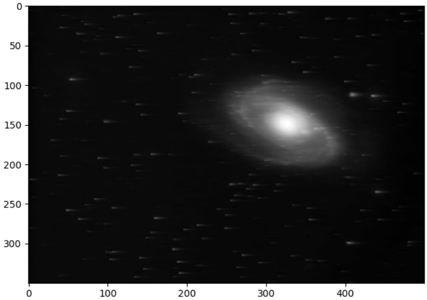


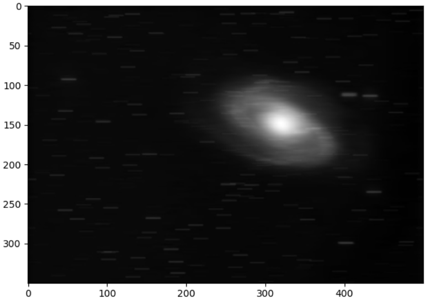
`scipy.ndimage.filters.gaussian_filter()` (nicht `scipy.signal.colvolve2d()`)

Verschiedene Kernel führen zu unterschiedlichen Effekten.
Beispiel: **Blur**



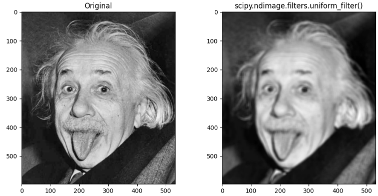
Comet-Kernel/Belichtungskernel



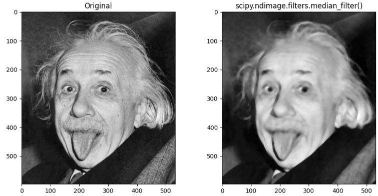


`scipy.ndimage.filters`

`uniform_filter()`: 2D-Mittelung mit fester Wichtung



`median_filter()`: 2D-Median mit fester Wichtung



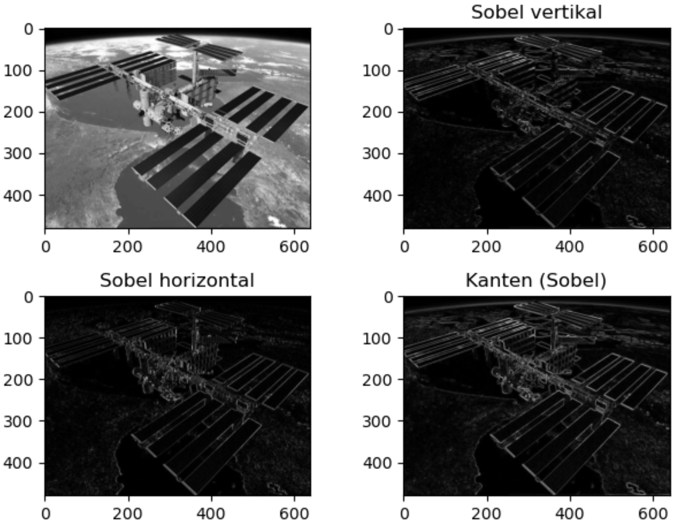
$$(f * g)_n = \sum_k^N f_k g_{n-k}$$

Kernel $(1; -1) \equiv (f * g)_n = f_n - f_{n-1}$: Vorwärtsableitung
Kernel $(\frac{1}{2}; 0; -\frac{1}{2}) \equiv (f * g)_n = \frac{1}{2}(f_{n+1} - f_{n-1})$: Zentrale Ableitung
Kernel $(1; -2; 1) \equiv (f * g)_n = f_{n+1} - 2f_n + f_{n-1})$: 2. zentr. Abl.
Kernel $(\frac{1}{3}; \frac{1}{3}; \frac{1}{3}) \equiv (f * g)_n = \frac{1}{3}(f_{n+1} + f_n + f_{n-1})$:
Mittelung/Glättung (*boxcar*)

Kantenerkennung im Bild: 2D Faltung mit Kernel $(1; -1)$ bzw. $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ oder besser **Sobelfilter** (Ableitung + Glättung)

$$S = \sqrt{S_x^2 + S_y^2}, S_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}, S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Sobelfilter (`scipy.ndimage.sobel()`):



Signalverarbeitung und -analyse: Dekonvolution

$$S = B * K$$

$$\mathcal{F}\{S\} = \mathcal{F}\{B\} \cdot \mathcal{F}\{K\}$$

$$\boxed{S = \mathcal{F}^{-1}(\mathcal{F}\{B\} \cdot \mathcal{F}\{K\})}$$

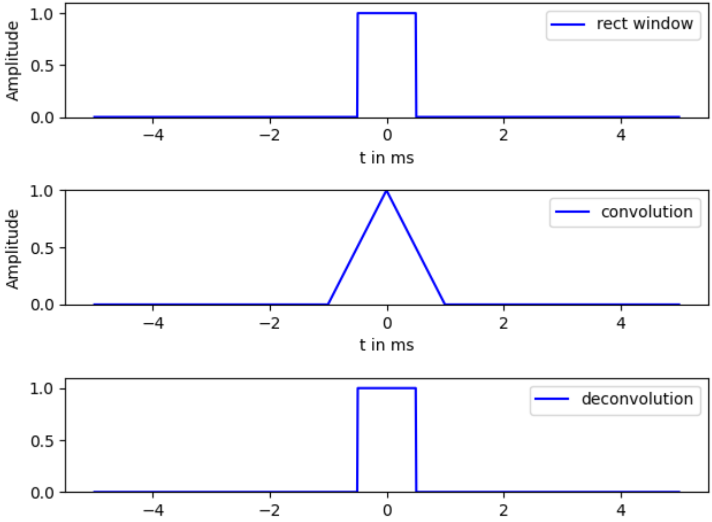
Umkehrung:

$$\mathcal{F}\{B\} = \frac{\mathcal{F}\{S\}}{\mathcal{F}\{K\}}$$

$$\boxed{B = \mathcal{F}^{-1}\left\{\frac{\mathcal{F}\{S\}}{\mathcal{F}\{K\}}\right\}}.$$

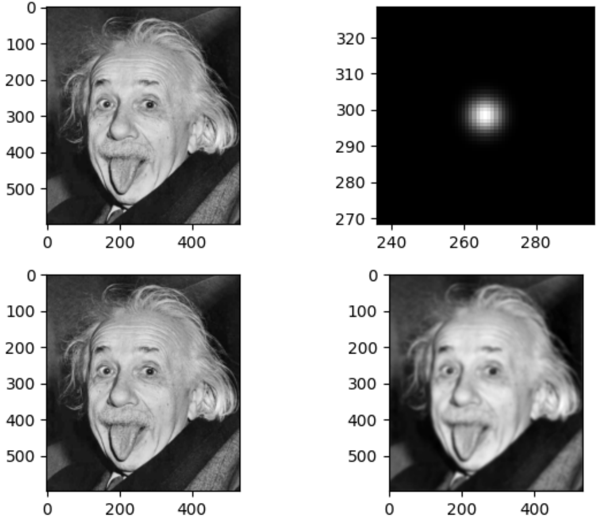
Dekonvolution: Zurückrechnen des Originalbildes bei bekanntem/geratenem Kernel

Pulsrekonstruktion:



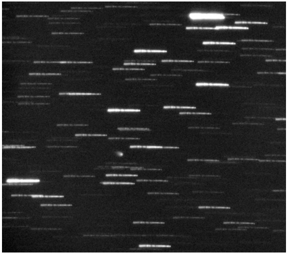
Signalverarbeitung und -analyse: Rekonstruktion

Bildrekonstruktion (Hier: Unschärfe):

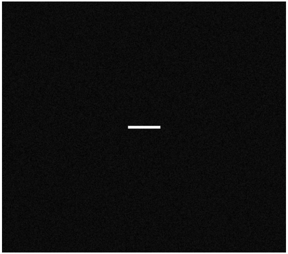


Beispiel: Bewegungskorrektur (s. Übung)

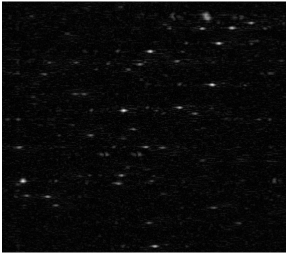
Sternspuren



Kernel



Dekonvolution



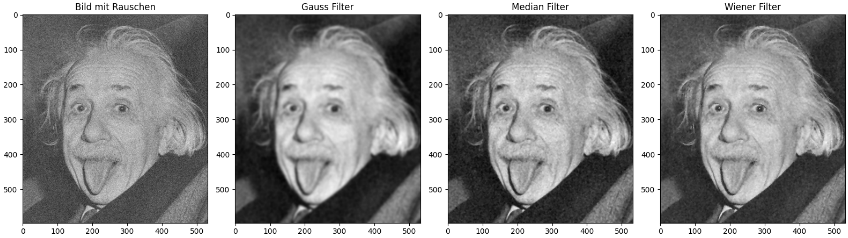
Signalverarbeitung und -analyse: Dekonvolution

$$\boxed{B = \mathcal{F}^{-1}\left\{\frac{\mathcal{F}\{S\}}{\mathcal{F}\{K\}}\right\}}.$$

Problem: Kernel K ist glatt, hat also keinen Hochfrequenzanteil. Damit Rauschen des Signals S wird extrem verstärkt!

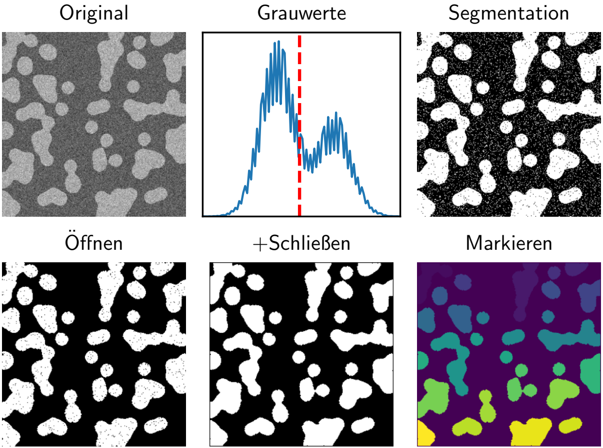
Lösung: **Wiener-Filter**, Optimale Rauschunterdrückung durch Anwendung der Methode der kleinsten Quadrate auf ein Signal mit additivem Rauschen um den Kernel K zu optmieren.

Filtervergleich:

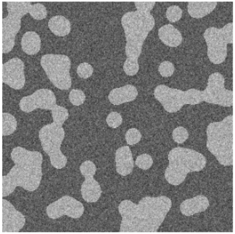


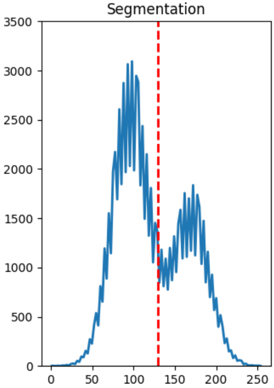
Signalverarbeitung und -analyse: Bildanalyse

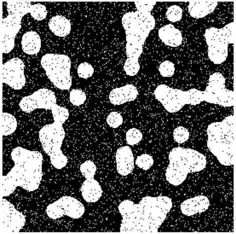
Informationen aus Bildern extrahieren. Beispiel: Segmentation, Morphologie und Granulometrie
$\rightarrow$ s. Buch Kap. 18.3.4



Segmentation: Trennung der Helligkeitswerte





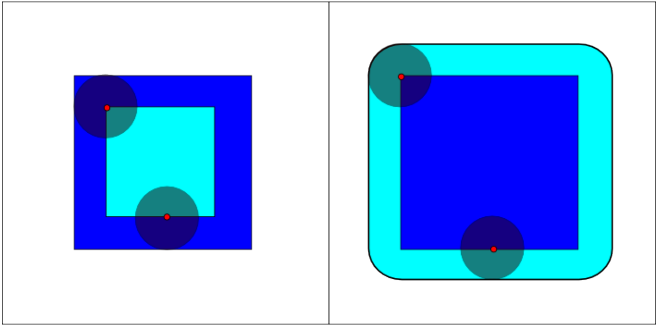


```
S = asarray(Image.open('ising.png').convert('L'))
BS = S > LIMIT
```

$\rightarrow$ *Morphing*

Erosion: $A \ominus B$: Alle Pixel, für die B in A passt.

Dilation: $A \oplus B$: B an jedem Pixel von A einfügen.

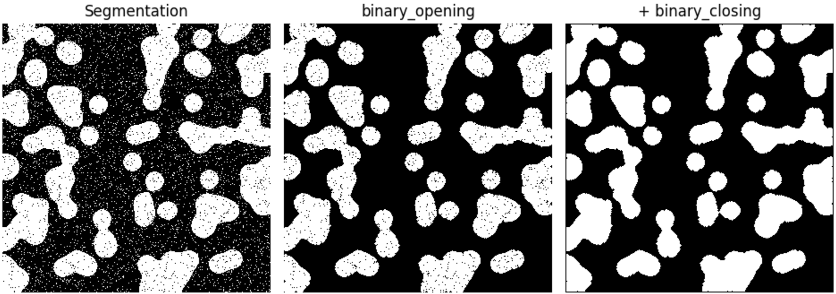


Binary Opening: $A \circ B = (A \ominus B) \oplus B$

Binary Closing: $A \bullet B = (A \oplus B) \ominus B$

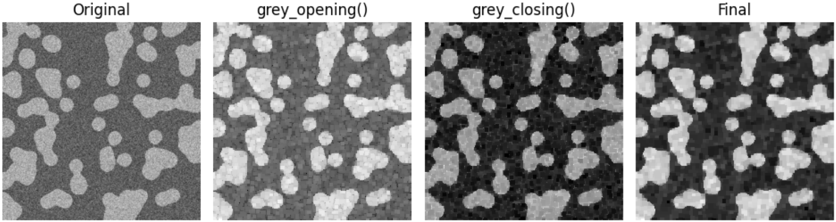
Binary Opening: $A \circ B = (A \ominus B) \oplus B$
Binary Closing: $A \bullet B = (A \oplus B) \ominus B$

```
scipy.ndimage.binary_opening(),
scipy.ndimage.binary_closing()
```



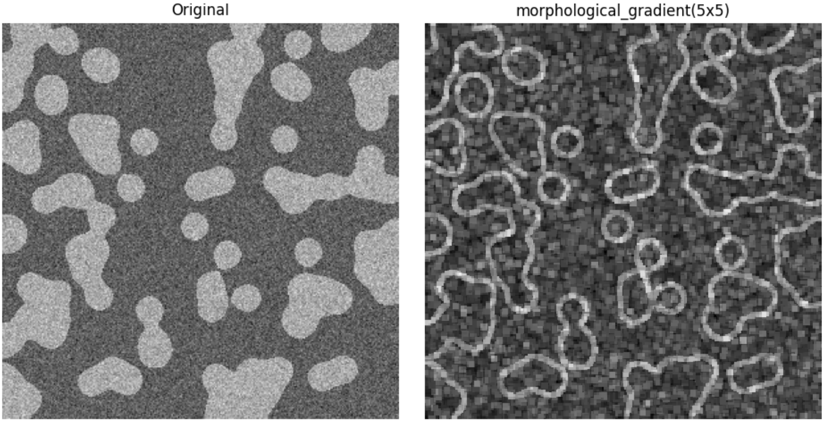
Grayscale Morphing: Maximum/Minimum Pixelwert in B

```
scipy.ndimage.grey_opening(),
scipy.ndimage.grey_closing()
```



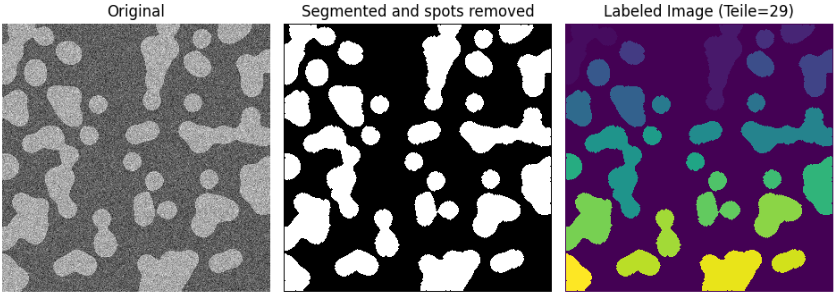
Gradient Morphing: $A \oplus B - A \ominus B$

`scipy.ndimage.morphology_gradient()`



Signalverarbeitung und -analyse: Labeling

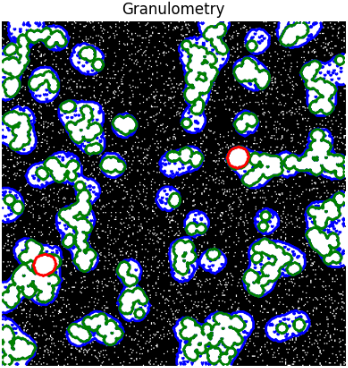
```
LI, nr_labels = scipy.ndimage.label(MS)
pl.imshow(LI, interpolation='nearest')
```

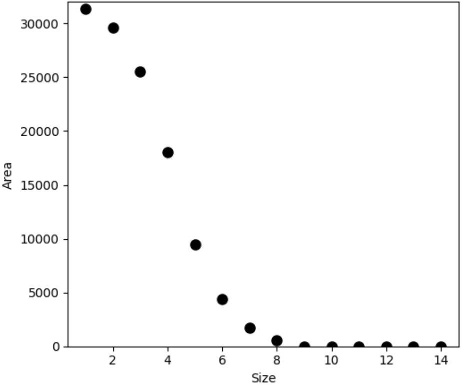


Signalverarbeitung und -analyse: Granulometrie

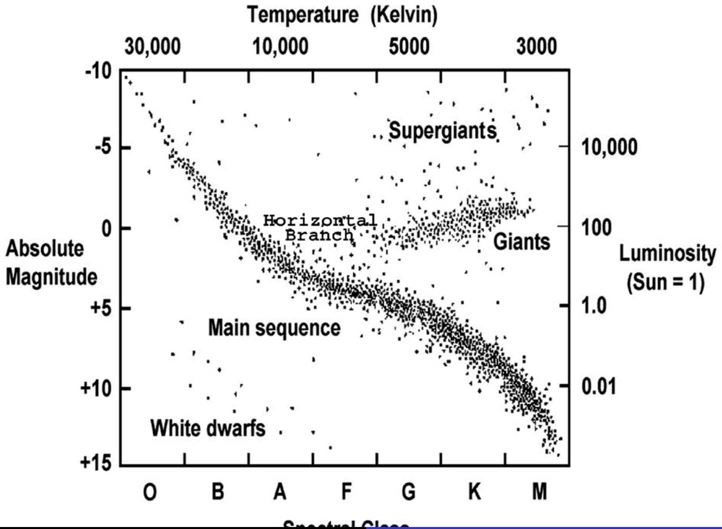
Bestimme die Größe/Größenverteilung von Gebieten:

```
def granulometry(image, sizes):
    return [binary_opening(image,
        structure=disk_structure(n)).sum() for n in sizes]

GS = granulometry(BS, sizes=arange(1, 15, 1))
```

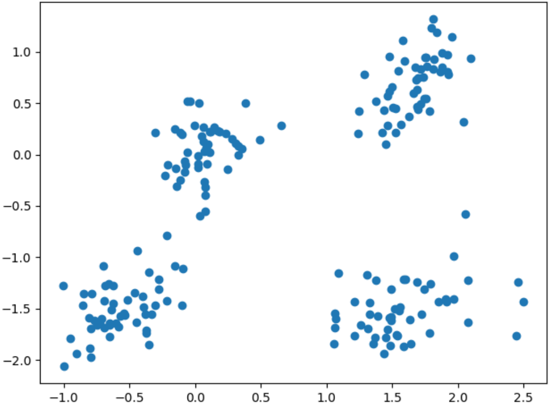




Finden/Trennen von Gruppen
Beispiel:



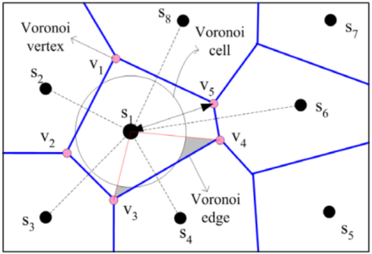
Beispiel:



k-means: Bilde k Gruppen aus Elementen durch Minimierung der Summe der quadr. Abweichungen von den k Zentren.

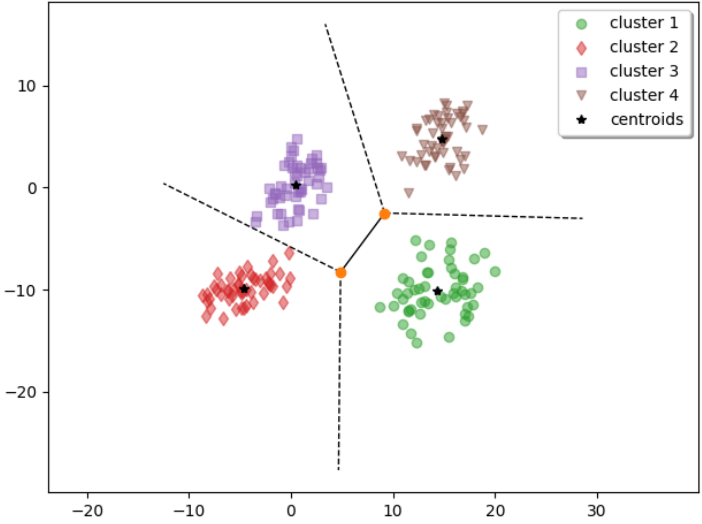
Signalverarbeitung und -analyse: Clusteranalyse

Voronoi-Diagramm: Zerlegung des Raumes anhand von Zentren. Region enthält alle Punkte, deren Abstand zu den anderen Zentren größer ist.

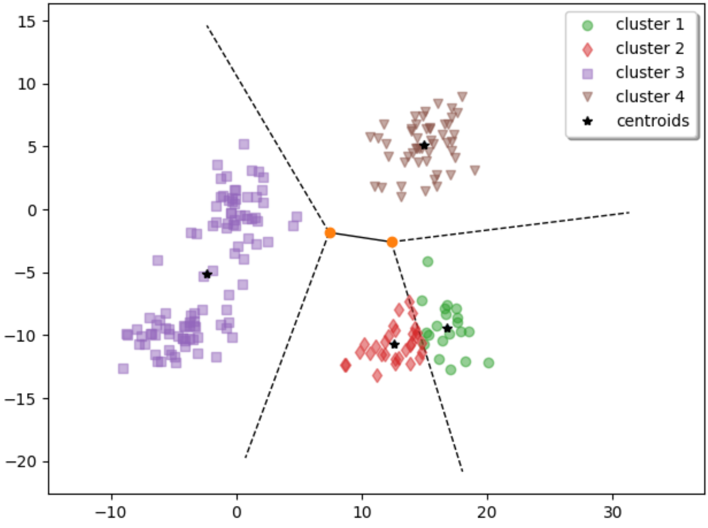



```python
from scipy.cluster.vq import kmeans2
centroid, label = kmeans2(data, 4, minit='points')

print("Counts: ", np.bincount(label))
pl.plot(centroid[:, 0], centroid[:, 1], 'k*', label='centroids')

from scipy.spatial import Voronoi, voronoi_plot_2d
vor = Voronoi(centroid)
fig = voronoi_plot_2d(vor, label='Voronoi')
```

Gutes Ergebnis:



Weniger gutes Ergebnis:



Korrelation:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f^*(\tau) g(\tau + t)\,\mathrm{d}\tau$$

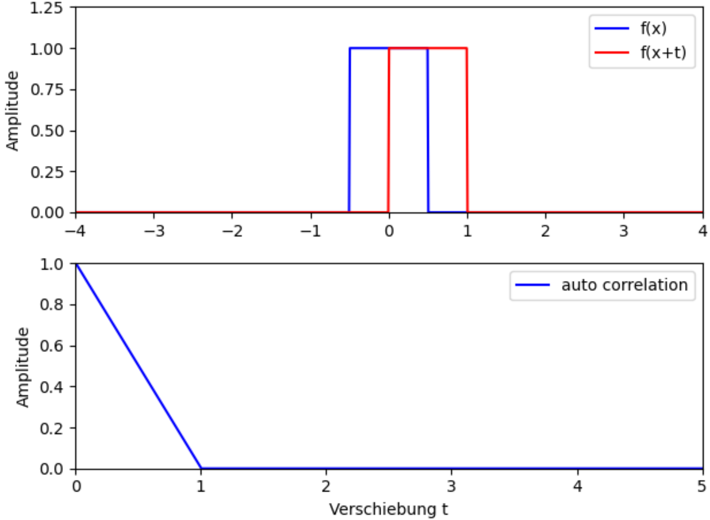
$$(f \star g)_j = \sum_k f_k^* g_{k+j}.$$

Gleitender Mittelwert von f mit Gewichtung/Muster g. Hohe Werte bei Übereinstimmung: **Mustererkennung**
$\mathcal{F}\{f \star g\} = k\mathcal{F}\{f^*(-t)\} \cdot \mathcal{F}\{g\} = k\mathcal{F}^*\{f\} \cdot \mathcal{F}\{g\}$.
Autokorrelation: Korrelation mit sich selbst: Erkennung von periodischen Mustern

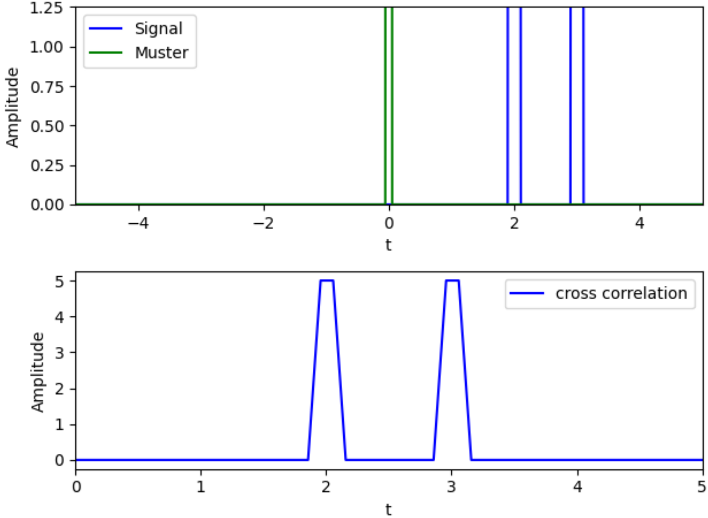
`scipy.signal.correlate()`, `scipy.signal.correlate2d()`

Signalverarbeitung und -analyse: Autokorrelation

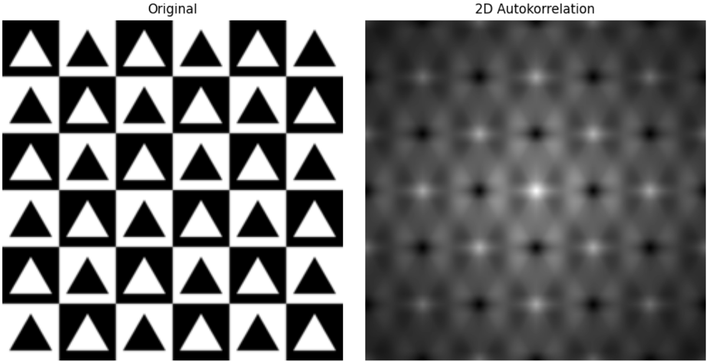
Übereinstimmung mit verschobener Kopie:



Übereinstimmung mit Muster:

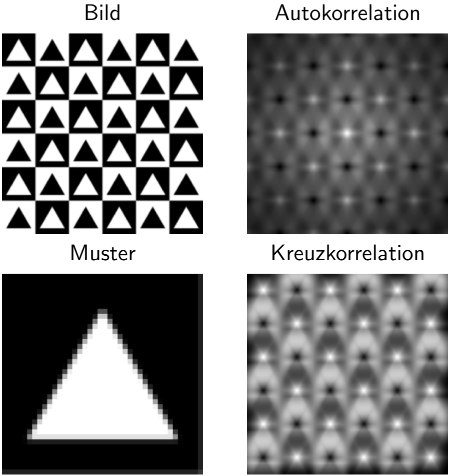


Periodische Muster im Bild: Autokorrelation
`scipy.signal.correlate(S,S)` (nicht `correlate2d()`!)



Mustersuche im Bild: Kreuzkorrelation

`scipy.signal.correlate(S,M)`



Signalverarbeitung und -analyse: Bildanalyse

Anwendung: Mustererkennung durch Kreuzkorrelation