TU/e Technische Universiteit Eindhoven University of Technology

Fourier analysis

prof.dr.ir.Bart M. ter Haar Romeny

```
SetOptions[ListPlot, PlotRange → All, Joined → True, AspectRatio → 0.4];
SetOptions[ArrayPlot, ColorFunction → GrayLevel, Frame → False];
```

1. Fourier Series

Fourier analysis is the mathematical description of periodic signals as sums of Cos and Sin functions. It is one of the most popular and useful theories for the analysis of **signals**, such as acoustical sounds in 1D and images in 2D.

The basic reason for its usefulness is that a temporal (or spatial) periodic signal can be transformed in a "Fourier series", and that it is often easier to do signal operations like filtering or even differentiation on this transformed signal. Let us look at an arbitrary sum of Sin and Cos functions:

```
Plot[Sin[t] + 0.37 Sin[2 t] + 0.23 Sin[3 t] + 0.74 Sin[5 t] +
  0.4 Cos[t] + 0.59 Cos[2 t] + 0.11 Cos[4 t], {t, 0, 14 π}, AspectRatio → 0.3]
```

This is a periodic function, and because we took the timescale from 0 to 14π, we get 7 periods of the lowest frequency (Sin[t]). This is the *base frequency*. The higher frequencies, which are multiples of the base frequency, are called the *harmonics*. We can make an infinite number of periodic functions this way. The amplitude factors in front of the Sin (1, 0.37, 0.23, 0, 0.74) and Cos (0.4, 0.59, 0, 1.11, 0) functions are called the *Fourier coefficients*.

```
Manipulate[
 Plot[Sin[t] + a Sin[2 t] + b Sin[3 t] + c Sin[5 t] + d Cos[t] + e Cos[2 t] + f Cos[4 t],
  {t, 0, 14 π}, AspectRatio → 0.3, PlotRange → {-2, 2}],
 {{a, 0.37}, -1, 1}, {{b, 0.23}, -1, 1}, {{c, 0.74}, -1, 1},
 {{d, 0.4}, -1, 1}, {{e, 0.59}, -1, 1}, {{f, 0.11}, -1, 1}]
```

```
Manipulate[Play[Sin[1000 t] + a Sin[2000 t] + b Sin[3000 t] +
   c Sin[5000 t] + d Cos[1000 t] + e Cos[2000 t] + f Cos[4000 t], {t, 0, 2}],
 {{a, 0.37}, -1, 1}, {{b, 0.23}, -1, 1}, {{c, 0.74}, -1, 1},
 {{d, 0.4}, -1, 1}, {{e, 0.59}, -1, 1}, {{f, 0.11}, -1, 1}]
```

It also works the other way: we can make from a given periodic signal the Sin and Cos series: This was first found by the Frenchman Jean-Baptiste Fourier (1822): "Every periodic function can be written as a series of Cos and Sin fuctions with increasing frequency":

$$f[t] = a_0 + \sum_{k=1}^{\infty} a_k \, \text{Cos}[k\,t] + \sum_{l=1}^{\infty} b_l \, \text{Sin}[l\,t]$$

with a_0 the average value, a_k the Fourier coefficient of the Cos series, and b_l the Fourier coefficient of the Sin series.

Jean-Baptiste Joseph Fourier (1768 - 1830).

Fourier proved that the Fourier coefficients can be calculated from the signal with the following formula's (which are discussed and proved in section 10.4.3 in the math section of the reader):

$$a_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f[t] \, dt;$$

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f[t] \, \text{Cos}[k\,t] \, dt;$$

$$b_l = \frac{1}{\pi} \int_{-\pi}^{\pi} f[t] \, \text{Sin}[l\,t] \, dt;$$

In 10.4.5 examples are given how to calculate a_k and b_l for some easy periodic functions, i.e. functions that can be easily described such as by straight lines (f[x]=$\frac{|x|}{\pi}$ → a triangle, f[x]= -1 and 1 → a block function). Only for such easy functions we are able to calculate the integrals for a_k and b_l analytically, i.e. by hand.

Every sinusoidal function is fully described by its amplitude, frequency and its phase. The graph of the amplitudes as a function of frequency is the *amplitude spectrum*, the graph of the phases as a function of frequency is the *phase spectrum.*

E.g. the sawtooth function with period p seconds (so the frequency is 1/p Hz) and amplitude a (given here with a base frequency and 3 harmonics, but in fact there are an infinite number of harmonics necessary to make the true sawtooth function):

```
sawtooth[t_] := a/2 + 2 a/π (Sin[2 π t/p] - 1/2 Sin[4 π t/p] + 1/3 Sin[6 π t/p] - 1/4 Sin[8 π t/p]);
```

```
sawtooth[t_, n_] := a/2 + 2 a/π Sum[(-1)^(m+1) 1/m Sin[2 m π t/p], {m, 1, n}]
```

We see that the truncated series with only 4 frequencies does only approximate the sawtooth function:

```
a = 1;
p = 2;
Plot[sawtooth[t], {t, 0, 10}, AspectRatio → 0.3, PlotPoints → 200]
```

```
a = 1;
p = 2;
Plot[sawtooth[t, 10], {t, 0, 10}, AspectRatio → 0.3, PlotPoints → 200]
```

When we plot the base frequency (and the average) also in the graph, we see the basic periodicity:

```
a = 1;
p = 2;
Plot[{sawtooth[t], a/2 + (2 a) Sin[2 π t/p]/π}, {t, 0, 10}, AspectRatio → 0.3]
```

The average is a/2 = 1/2 and we can see this in the graph. We can also say that this is the Fourier coefficient of the lowest possible frequency, which is zero. Zero frequency is a signal that does not fluctuate at all, in other words: a constant signal.

So we have the frequencies {0, $\frac{1}{p}$, $\frac{2}{p}$, $\frac{3}{p}$, $\frac{4}{p}$} and at these frequencies we have the amplitudes $\left\{\frac{a}{2}, \frac{2a}{\pi}, \frac{-a}{\pi}, \frac{2a}{3\pi}, \frac{-a}{2\pi}\right\}$ and the phases {0, 0, π, 0, π}. We define the phase of the average to be 0. We plot the amplitude and phase spectra of this function:

```
ampl = ListPlot[{{0, a/2}, {1/p, 2 a/π}, {2/p, -a/π}, {3/p, 2 a/(3 π)}, {4/p, -a/(2 π)}},
  PlotStyle → PointSize[0.03], Filling → Axis, Joined → False, AspectRatio → 0.5]
```

```
phase = ListPlot[{{0, 0}, {1/p, 0}, {2/p, π}, {3/p, 0}, {4/p, π}},
  PlotStyle → PointSize[0.03], Filling → Axis, Joined → False, AspectRatio → 0.5];
GraphicsGrid[{{ampl, phase}}]
```

Signals with fast (steep) variations are build from high frequency wave functions, and vice versa.

Filtering is cutting out frequencies from the spectrum.
E.g. when we want to cancel low frequency noise (e.g. slow variations in an EEG signal) we apply a *high-*

pass filter. Other slow variation noise sources are 50 Hz noise and, in images, light variations due to overcast shadows.
When we want to reduce high frequency noise (e.g. speckle noise in images) we apply a *low-pass filter*.

In the Fourier domain a filter can be very simple. For example: a stepfunction which is zero for 0 < x < 1.75, and one for x > 1.75. When we **multiply** the spectrum with this filter, we make the Fourier coefficients for the frequencies {0, $\frac{1}{2}$, 1, $\frac{3}{2}$} zero and the frequency 2 is the only one left after this high-pass filtering.

Spatial frequencies are variations over space. The wavelength is expressed in meters, the frequency in cycles per meter. Here is an example of a sinusoidal spatial function with increasing frequency in the x-direction:

```
ArrayPlot[Table[Sin[t^2.2], {y, 0, 10}, {t, 0, 4 π, 0.01}], AspectRatio → 0.2]
```

2. Examples of Fourier series

```
Off[General::"spell1"];
Off[NIntegrate::"ploss"];
Off[NIntegrate::"ncvb"];
Off[NIntegrate::"inum"];
Off[NIntegrate::"slwcon"];
Off[NIntegrate::"oidiv"];
```

To make even functions with period `2L`, we can start with the zigzag function.

```
zigzag[x_] := 1 - 2 Abs[x / (2 L) - 1 / 2 - Floor[x / (2 L)]]
```

Its graph looks like a zigzag.

```
L = 1;
Plot[zigzag[x], {x, -5, 5}, PlotStyle -> {{Green, Thickness[.01]}}]
```

Functions defined in terms of the zigzag function are even and periodic.

```
Plot[ Tan[zigzag[x]], {x, -2, 2}, PlotStyle -> {{Red, Thickness[.01]}}]
```

This defines a block function with amplitude `amp` and period `2L`:

```
f[x_] := If[zigzag[x] < 1/2, amp, 0]
```

Have a look, with rather arbitrary choices for `amp` and `L`:

```
amp = 3 / 4; L = 1.4;
Plot[f[x], {x, -3, 3}, PlotPoints -> 100,
    PlotStyle -> {Red, Thickness[0.01]}]
```

The Fourier coefficients of this even function f are defined by:

```
Clear[a];
a[0] = 1/(2 L) NIntegrate[f[t], {t, -L, L}];
a[n_] := a[n] =
1/L NIntegrate[f[t] Cos[π n t/L], {t, -L, L}];
b[l_] := b[l] =
1/L NIntegrate[f[t] Sin[π l t/L], {t, -L, L}];
```

```
b[2]
```

```
b[4]
```

The first few Fourier coefficients:

```
ListPlot[Table[{n, a[n]}, {n, 0, 55}], PlotRange -> All,
 Joined → False, PlotStyle -> {Red, PointSize[.02]}]
```

Because the function is symmetric around zero, the Sin components are all zero:

```
ListPlot[Table[{l, b[l]}, {l, 0, 55}],
 PlotRange -> All, PlotStyle -> {Blue, PointSize[.01]}]
```

We show a few approximations of f by partial sums of its Fourier series.

```
plottot[grens_, opt___] := (approx[x_] = Sum[a[n] Cos[π n x/L], {n, 0, grens}];
   Plot[{f[x], approx[x]}, {x, -2, 2}, opt]);
```

```
Manipulate[plottot[n, PlotPoints → 200, PlotRange → {-0.25 amp, 1.2 amp},
  PlotStyle → {{Red, Thickness[0.01]}, {Green, Thickness[0.01`]}}], {n, 1, 20, 1}]
```

```
sum55 = plottot[55, PlotPoints -> 200]
```

The Gibbs phenomenon in closeup:

```
Show[sum55, PlotRange -> {{0, 2}, {.6, .86}}]
```

Fourier applet

3. The Fourier transform

Fourier series

$$f[t] = a_0 + \sum_{k=1}^{\infty} a_k \, \text{Cos}[k\,t] + \sum_{l=1}^{\infty} b_l \, \text{Sin}[l\,t]$$

$$a_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f[t] \, dt;$$

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f[t] \, \text{Cos}[k\,t] \, dt;$$

$$b_l = \frac{1}{\pi} \int_{-\pi}^{\pi} f[t] \, \text{Sin}[l\,t] \, dt;$$

Fourier transform

$$\text{ExpToTrig}\left[e^{i\,\phi}\right]$$

The Fourier transform $F(\omega)$ of $f(x)$ is by definition

$$F(\omega) = \int_{-\infty}^{\infty} f(x) \, e^{-i\omega x} \, dx$$

The Inverse Fourier transform $f(x)$ of $F(\omega)$ is by definition

$$f(x) = \int_{-\infty}^{\infty} F(\omega) \, e^{i\omega x} \, dx$$

4. Convolution is a product in the Fourier domain: the convolution theorem.

It is often not so easy to calculate the convolution integral. It is often much easier to calculate the integral of the convolution in the Fourier domain. This section explains how this works. We start from the convolution of the function $h(x)$ with the kernel $g(x)$:

$$f(x) = \int_{-\infty}^{\infty} h(y) \, g(x-y) \, dy$$

The Fourier transform $F(\omega)$ of $f(x)$ is then by definition

$$F(\omega) = \int_{-\infty}^{\infty} f(x) \, e^{-i\omega x} \, dx = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(y) \, g(x-y) \, dy \, e^{-i\omega x} \, dx$$

We reorder the terms in the integral and make the substitution $x - y = \tau$, so we have $x = \tau + y$ and $e^{-i\omega x} = e^{-i\omega\tau} . e^{-i\omega y}$:

$$\int_{-\infty}^{\infty} h(y) \int_{-\infty}^{\infty} e^{-i\omega x} g(x-y)\, dx\, dy = \int_{-\infty}^{\infty} h(y) \int_{-\infty}^{\infty} e^{-i\omega(\tau+y)} g(\tau)\, d(\tau+y)\, dy$$

When we integrate to τ, we keep y constant, so $d(\tau+y)$ is equal to $d\tau$. So we get:

$$\int_{-\infty}^{\infty} h(y) \int_{-\infty}^{\infty} e^{-i\omega\tau} e^{-i\omega y} g(\tau)\, d\tau\, dy$$

Because $e^{-i\omega y}$ is constant in the integration to τ, we may bring it as a constant outside the integral:

$$\int_{-\infty}^{\infty} h(y)\, e^{-i\omega y}\, dy \int_{-\infty}^{\infty} e^{-i\omega\tau} g(\tau)\, d\tau = H(\omega) . G(\omega)$$

where $H(\omega)$ is the Fourier transform of the function $h(x)$ and $G(\omega)$ is the Fourier transform of the kernel $g(x)$. So the Fourier transform of a convolution is the product of the Fourier transform of the filter with the Fourier transform of the signal. To put it otherwise:

$$\begin{array}{ccccc} f(x) & = & h(x) \otimes g(x) \\ \updownarrow & & \updownarrow & & \updownarrow \\ F(\omega) & = & H(\omega) . G(\omega) \end{array}$$

We can calculate a convolution $h \otimes g$ by calculating the Fourier transforms $H(\omega)$ and $G(\omega)$, multiply them to get $F(\omega)$, and we get $f(x)$ by taking the inverse Fourier transform of $F(\omega)$. Often this is a much faster method then calculating the convolution integral, because the routines that calculate the Fourier transform are so fast. We look at an example where we filter noise from the data. We simulate a signal of a noisy ECG recording:

```
data = Table[N[Sin[2 π n / 128] + 0.8 (Random[] - 1/2)], {n, 1, 256}];
ListPlot[data, Joined → True, AspectRatio → 0.4`]
```

This generates a typical convolution kernel suitable for smoothing data.

```
σ = 10;

kernel = Table[e^(-n^2/(2 σ^2)) / Sqrt[2 π σ^2], {n, -128, 127}]; ListPlot[kernel]
```

This puts the convolution kernel in a form suitable for use in a discrete Fourier transform (we have to bring it to the origin):

```
kernel = RotateLeft[kernel, 128];
ListPlot[kernel]
```

The Fourier transform of the data looks like this (a very low frequency and at the higher frequencies the noise):

```
p1 = ListPlot[Abs[Fourier[data]]]
```

The Fourier transform of the kernel looks like a low-pass filter:

```
p2 = ListPlot[128 Abs[Fourier[kernel]], PlotStyle → Red]
```

```
Show[{p1, p2}]
```

```
Show[{p1, p2}, PlotRange → {{0, 30}, All}]
```

```
ListPlot[Abs[Fourier[kernel] × Fourier[data]]]
```

See how we cleaned up the spectrum. We are now "in the Fourier domain", and we can go back to our temporal domain by means of the inverse Fourier transform.

The convolution is done by multiplying the Fourier transform of the data by the Fourier transform of the kernel, then taking the inverse Fourier transform of the result.

```
conv = InverseFourier[Sqrt[256] Fourier[data] Fourier[kernel] ] ;
```

This plots the result. The Chop removes small imaginary parts that are generated in the Fourier transforms. The final plot is a *smoothed* version of the original data.

```
ListPlot[Chop[conv]]
```

Temporal versus frequency domain: There is always the inverse relation: when a wavelength is *small* in the temporal or spatial domain, it represents a *high* frequency. A low frequency represents a long wavelength, with a large representation in the temporal/spatial domain. The shortest function, the *spike- or delta-function*, has all frequencies at aplitude one, with phase zero. The longest (infinite) wavelength has frequency zero.

Removing the high frequencies from a signal (low-pass filtering, blurring) makes the signal less spiky, more smooth. Removing the low frequencies from a signal also removes the average component (the average becomes zero), and slow trends and variations are cancelled.

White noise has a spectrum with all frequencies at amplitude one, and a random phase for each frequency. *Pink noise* has only amplitudes at a limited band of frequencies. *Band-limited noise* misses the high or the low frequencies.

If a function is not periodical, is can be made periodical by simple repetition left and right of the original function, from minus infinity to plus infinity. In the spatial domain this means tiling of the whole space with the 2D function.

5. A linear system

The system described by a convolution integral is called a *linear system*. A linear system is characterised by two properties:

$f[t] \otimes h[t] = g[t] \Rightarrow$
1. Superposition: $(f_1[t] + f_2[t]) \otimes h[t] = f_1[t] \otimes h[t] + f_2[t] \otimes h[t] = g_1[t] + g_2[t]$
2. Homogeneity: $a\, f[t] \otimes h[t] = a\, g[t]$

E.g. the process of projecting an external line function on the retina is described by such a convolution. The input function f[x,y] is a very narrow line, the output function g[u,v] is the blurred image on the retina, and the kernel h[x,y] is the blurring function, also called the *line-spread function (LSF)*. When a point is projected, we call h[x,y] the *point-spread function (PSF)*.

When all elements of a shifting template are equal and their sum is 1, the filter is called a *running average filter*. When e.g. the length is 5 (the filter looks like {1/5,1/5,1/5,1/5,1/5} or 1/5{1,1,1,1,1}), it takes the average over 5 samples at each shift position. This is a lowpass filter.

6. The response on a spike function: the Point Spread Function (PSF)

We saw earlier that the convolution of a kernel with the delta (spike) function gives the kernel itself:

■ The δ-function (1D point impulse)

The smallest possible function, with area 1, is the delta function, also known as the Dirac function, or the impulse function. This function is defined by:

$$\delta[x] = 0,\ x \neq 0$$
$$\int_{-\infty}^{\infty} f[x]\, \delta[x]\, dx = f[0]$$

In *Mathematica*, this function can by called by DirecDelta.

```
Clear[f]
```

$$\int_{-\infty}^{\infty} f[x]\, \text{DiracDelta}[x]\, dx$$

An important property is the selection of a specific value of f[x] on the x-axis, when we shift the delta function to that position. This is the the *sifting* property of the delta function:

$$\text{Assuming}\left[a \in \text{Reals},\ \int_{-\infty}^{\infty} f[x]\, \text{DiracDelta}[x-a]\, dx\right]$$

Suppose we like to see the shape of our convolution kernel. The solution is to take the spike image as input:

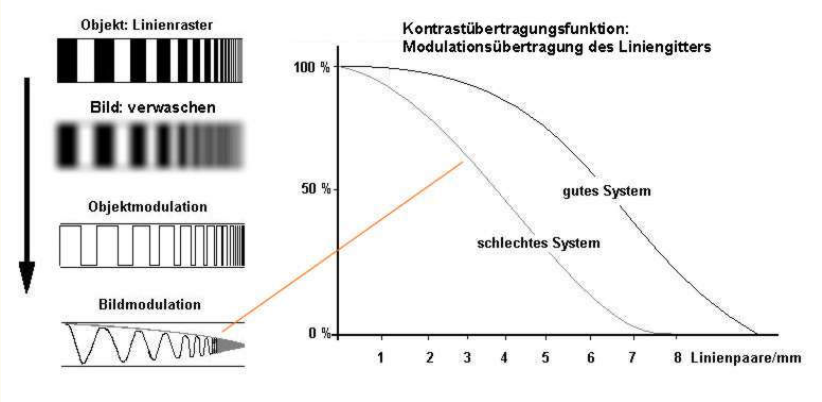
```
lettera = ImageData[
    Import["http://bmia.bmt.tue.nl/people/BRomeny/Courses/8C080/a.gif"]][[All, All, 1]];
```

```
spike = Table[0, {200}, {200}]; spike[[100, 100]] = 1;
ArrayPlot[spike]
```

```
ArrayPlot[ListConvolve[lettera, spike]]
```

If one wants to determine how good an imaging system is, one needs to know how a very tiny *point* is imaged. E.g. in a CT scanner one can make an image of a very thin (100 μm) copper wire. This functions as a delta funtion, and the image measured gives the Point Spread Funtion, which is a measure for the quality (spatial resolution). Recall the measurements on the human retina in the bginning.

The Fourier transform of the PSF is called the Modulation Transfer Function.



A frequently applied filter is the Gaussian function:

```
Unprotect[gauss];
```

PS: The function gauss[] is already defined in the package MathVisionTools, and protected. After 'Unprotect' we can define it anew.

```
gauss[x_] := 1/Sqrt[2 π σ^2] E^(-x^2/(2 σ^2))
```

Remarkably, the Fourier transform of a spatial Gaussian function, as a function of x, is also a Gaussian function, of course as a function of spatial frequency ω:

```
σ = .;
Simplify[FourierTransform[1/Sqrt[2 π σ^2] E^(-x^2/(2 σ^2)), x, ω], σ > 0]
```

So, we can do Gaussian blurring (for noise removal) in the spatial domain with the convolution with a Gaussian kernel, or in the frequency domain, with a Gaussian kernel!

Note that the σ is now as a multiplier, so a narrow Gaussian in the spatial domain is a wide Gaussian in the frequency domain, and vice versa.

```
Manipulate[
 GraphicsGrid[{{Plot[1/Sqrt[2 π σ^2] E^(-x^2/(2 σ^2)), {x, -5, 5}, PlotRange → {0, .8}, AxesLabel → {"x",},
     PlotLabel → "Spatial domain"], Plot[E^(-1/2 σ^2 ω^2)/Sqrt[2 π], {ω, -5, 5}, PlotRange → {0, 0.4},
     AxesLabel → {"ω",}, PlotLabel → "Frequency domain"]}}], {{σ, 1}, .1, 5}]
```

7. Spatial frequencies

Here is a spatial (spatial means: over space, along the x-axis, expressed in meters) signal with a single frequency (unit: cycles per meter):

```
ω = 1; Plot[Sin[2 π ω x], {x, 0, 4}]
```

This changes the spatial frequency:

```
Manipulate[Plot[Sin[2 π ω x], {x, 0, 4}, PlotRange → {-1, 1}], {{ω, 1}, 0, 4}]
```

Here is the frequency shown as a 2D signal, with a spatial frequency along the x-axis, and a constant frequency of zero cycles per meter in the y-direction:

```
ω = 1; spatfreq = Table[Sin[2 π ω x], {y, 0, 4, .01}, {x, 0, 4, .01}];
ArrayPlot[spatfreq]
```

```
Manipulate[spatfreq = Table[Sin[2 π ω x], {y, 0, 4, .01}, {x, 0, 4, .01}];
 ArrayPlot[spatfreq], {{ω, 1}, 0, 4}]
```

And here we change the spatial frequencies in the x- and y-direction:

```
Manipulate[
 spatfreq = Table[(1 + Sin[2 π ωx x]) (1 + Sin[2 π ωy y]), {y, 0, 4, .05}, {x, 0, 4, .05}];
 ArrayPlot[spatfreq, PlotRange → All], {{ωx, 1}, 0, 4}, {{ωy, 1}, 0, 4}]
```

```
Manipulate[
 spatfreq = Table[(1 + Sin[2 π ωx x]) (1 + Sin[2 π ωy y]), {y, 0, 4, .05}, {x, 0, 4, .05}];
 ListPlot3D[spatfreq, PlotRange → All], {{ωx, 1}, 0, 4}, {{ωy, 1}, 0, 4}]
```

The 2D Fourier transform and 2D spectrum

```
ωx = 1;
ωy = 2;
Manipulate[spatfreq = Table[Sin[2 π ωx x] Sin[2 π ωy y] + μ, {y, 0, 4, .1}, {x, 0, 4, .1}];
 GraphicsGrid[{{, ArrayPlot[spatfreq, PlotRange → {-1, 5}],},
   {ArrayPlot[Re[Fourier[spatfreq]], PlotRange → {0, 10}],
    ArrayPlot[Im[Fourier[spatfreq]], PlotRange → {0, 10}],
    ArrayPlot[Abs[Fourier[spatfreq]], PlotRange → {0, 10}]}}],
 {{μ, 1}, 0, 2}, {{ωx, 1}, 0, 4}, {{ωy, 1}, 0, 4}]
```

The spectrum of white noise is uniform (*all* frequencies are randomly present in the noise signal):

```
noise = Table[Random[] - .5, {128}];
ListPlot[Abs[Fourier[noise]], Joined → True, PlotRange → All]
```

The Fourier transform of a delta spike is constant, is has a *flat spectrum*.

```
FourierTransform[DiracDelta[x], x, ωx]
```

The spectrum of a point image is completey 'white', all frequencies are present:

```
ArrayPlot[s = Abs[Fourier[spike]], PlotRange → {0, 0.01}]
```

```
ListPlot3D[s]
```

8. Blurring in the Fourier domain

```
ArrayPlot[mr64 = ImageData[
    Import["http://bmia.bmt.tue.nl/Education/Courses/FEV/book/images/mr64.gif"]][[
   All, All, 1]]]
```

```
ArrayPlot[Abs[fouriermr64 = Fourier[mr64]], PlotRange → {0, 5}]
```

```
σ = 1;
gaussft = Table[E^(-1/2 σ^2 (ωx^2+ωy^2)) / Sqrt[2 π], {ωx, -3.15, 3.15, .1}, {ωy, -3.15, 3.15, .1}];
```

```
ArrayPlot[gaussft, AxesLabel → {"ωx", "ωy"}, Axes → True]
```

We shift the Gaussian also to the origin:

```
ArrayPlot[shiftedgaussft = RotateLeft[gaussft, Dimensions[gaussft]/2]]
```

Now we are ready to do the filtering in the Fourier domain:

```
ArrayPlot[Abs[InverseFourier[shiftedgaussft × fouriermr64]], PlotRange → All]
```

```
Manipulate[gaussft = Table[E^(-1/2 σ^2 (ωx^2+ωy^2))/Sqrt[2 π], {ωx, -3.15, 3.15, .1}, {ωy, -3.15, 3.15, .1}];
 shiftedgaussft = RotateLeft[gaussft, Dimensions[gaussft]/2];
 GraphicsGrid[{{ArrayPlot[shiftedgaussft], ArrayPlot[
    Abs[InverseFourier[shiftedgaussft × fouriermr64]], PlotRange → All]}}], {σ, .5, 8}]
```