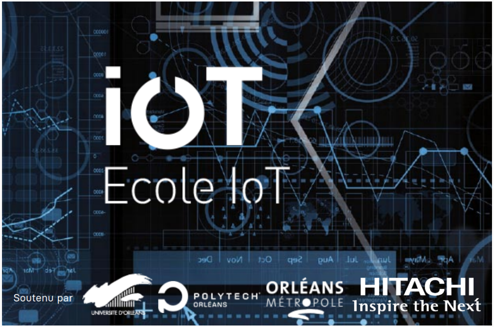
Ecole IoT: Data Science

Course III – Filtering in Fourier domain and variational methods for inverse problems

Bruno Galerne

Friday January 12, 2024



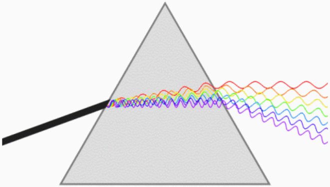
Basics of filtering

Standard filters

Two main approaches:

- **Spatial domain:** use the pixel grid / spatial neighborhoods
- **Spectral domain:** use Fourier transform, cosine transform, . . .

Spatial



Spectral

Spectral filtering

Spectral filtering – Periodical functions

A sine wave (or sinusoidal) $f(t) = a\cos(2\pi u t + \varphi)$ is periodical

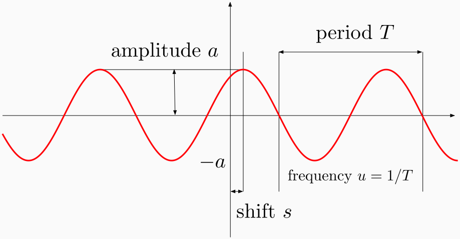
$$f(t+T) = f(t) \quad \text{for} \quad T = 1/u, \quad \text{for all} \quad t \in \mathbb{R}$$

and characterized by

- u : frequency ($u = 1/T$)
- a : amplitude
- φ : phase ($\varphi = -2\pi u s$)

where

- T : period
- s : shift



Spectral filtering – Discrete Fourier Transform (DFT)

Discrete signals

- Let $f \in \mathbb{R}^n$ be a discrete signal
- Consider it to be periodical: $f_{k+n} = f_k$
- It can be characterized only by its n harmonics of the form:

$$\frac{-\lceil n/2 \rceil + 1}{n}, \ldots, -\frac{2}{n}, -\frac{1}{n}, 0, \frac{1}{n}, \frac{2}{n}, \ldots, \frac{\lfloor n/2 \rfloor}{n}$$

- The discrete Fourier transforms (DFT) is thus given by

$$\underbrace{\hat{f}_u = \mathcal{F}[f]_u = \sum_{k=0}^{n-1} f_k e^{-i2\pi\frac{uk}{n}},}_{\text{Discrete Fourier transform}} \quad u = 0 \ldots n-1$$

and $$\underbrace{f_k = \mathcal{F}^{-1}[\hat{f}]_k = \frac{1}{n} \sum_{u=0}^{n-1} \hat{f}_u e^{i2\pi\frac{uk}{n}},}_{\text{inverse DFT}} \quad k = 0 \ldots n-1$$

Why does it matter? **It allows us to do signal processing**.

Spectral filtering – 2d DFT

Discrete images

- Let $f \in \mathbb{R}^{n_1 \times n_2}$ be a discrete image
- Consider it to be periodical: $f_{k+n_1,l+n_2} = f_{k,l}$
- The 2d discrete Fourier transforms (DFT) is thus given by

$$\underbrace{\hat{f}_{u,v} = \mathcal{F}[f]_{u,v} = \sum_{k=0}^{n_1-1} \sum_{l=0}^{n_2-1} f_{k,l} e^{-i2\pi\left(\frac{uk}{n_1}+\frac{vl}{n_2}\right)}}_{\text{2D DFT}}$$

and $$\underbrace{f_{k,l} = \mathcal{F}^{-1}[\hat{f}]_{k,l} = \frac{1}{n_1 n_2} \sum_{u=0}^{n_1-1} \sum_{v=0}^{n_2-1} \hat{f}_{u,v} e^{i2\pi\left(\frac{uk}{n_1}+\frac{vl}{n_2}\right)}}_{\text{inverse 2D DFT}}$$

- The pair (u, v) represents a two-dimensional frequency.

What does it look like?

- Each point (u, v) in the Fourier domain corresponds to a sine "wave" of frequency $\sqrt{u^2 + v^2}$ along the axis Δ directed by the vector (u, v)

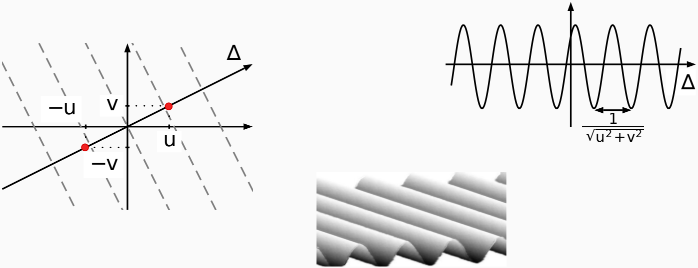


Figure 1 – 2D signals with spectrum limited only to frequencies (u, v) and $(-u, -v)$

$$= \frac{1}{n} \quad \hat{f}_{u_1,v_1} \cdot e^{i2\pi\left(\frac{u_1 k}{n_1}+\frac{v_1 l}{n_2}\right)}$$

$$+$$

$$\hat{f}_{u_1,v_2} \cdot e^{i2\pi\left(\frac{u_1 k}{n_1}+\frac{v_2 l}{n_2}\right)}$$

$$+$$

$$\hat{f}_{u_i,v_j} \cdot e^{i2\pi\left(\frac{u_i k}{n_1}+\frac{v_j k}{n_l}\right)}$$

$$\vdots$$

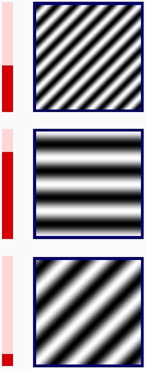
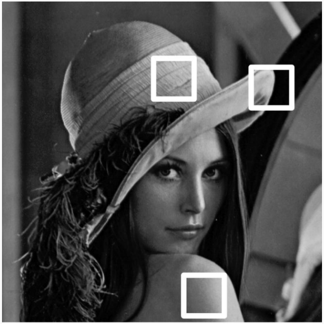


Image = weighted sum of sine waves

Spectral filtering – 2d DFT

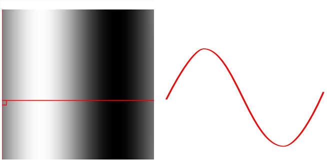
- In practice: all frequencies are more or less used in different regions



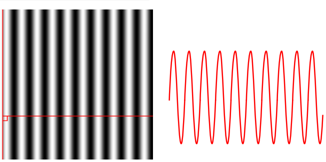
Which kinds of frequencies are used in the white squares?

Spectral filtering – 2d DFT

- Spatial frequency: measures how fast the image varies in a given direction



Low frequency (slow variations)

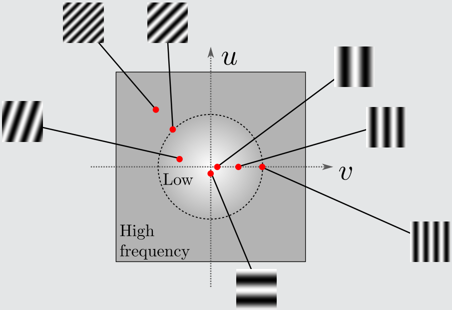


High frequency (fast variations)

How do we represent the Fourier coefficients?

Spectral filtering – 2d DFT

- Represent each Fourier coefficients on a 2d grid

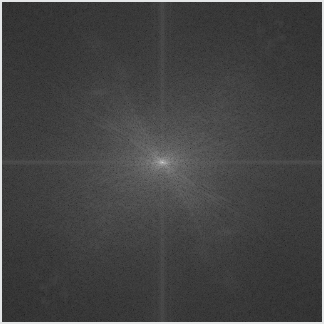


- $|\hat{f}_{u,v}|$: contribution of frequency $\sqrt{u^2 + v^2}$ in the direction (u, v).
- $\arg \hat{f}_{u,v}$: phase shift of frequency $\sqrt{u^2 + v^2}$ in the direction (u, v).
- Center $\equiv$ low frequencies
- Periphery $\equiv$ high frequencies

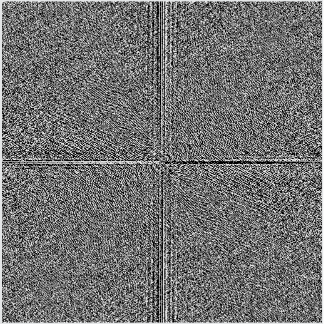
Spectral filtering – 2d DFT

Example

→



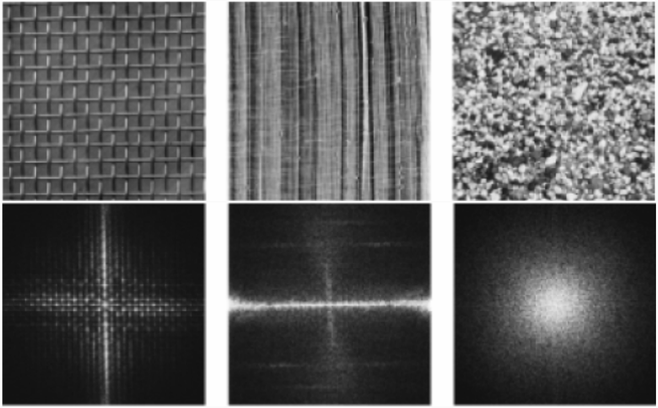
Amplitude



Phase

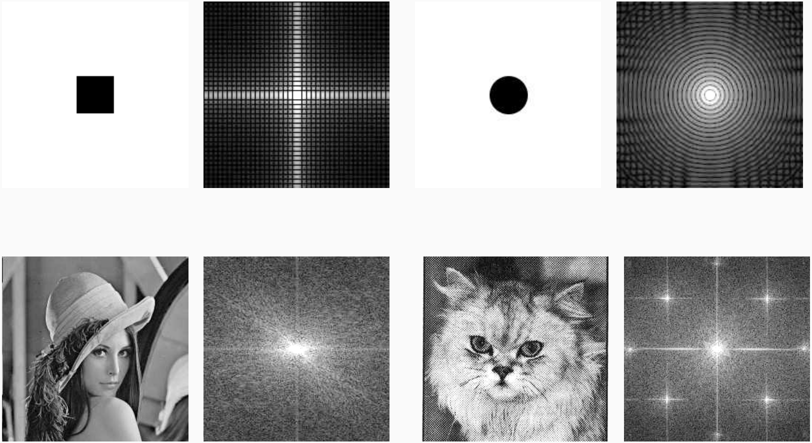
How to interpret it?

Spectral filtering – 2d DFT

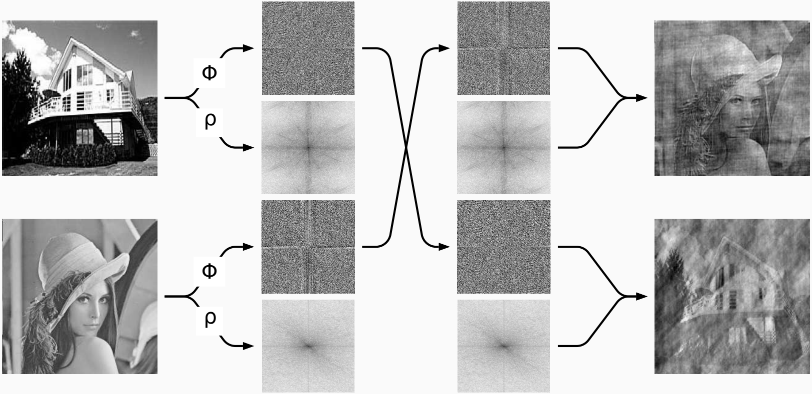


- Amplitude spectrum highlights the "directions" of a pattern
- Edge is represented by all harmonics in its orthogonal direction
- i.e., a line in the orthogonal direction (passing through the origin)

Spectral filtering – 2d DFT



Spectral filtering – 2d DFT



- In general, we only represent the modulus
- Nevertheless, the phase encodes a large amount of information

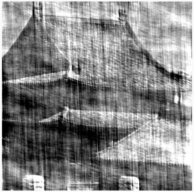
Modulus and phase of a digital image

Exchanging the modulus and the phase of two images:

Image 1

Image 2

Modulus of 1
& phase of 2



Modulus of 2
& phase of 1

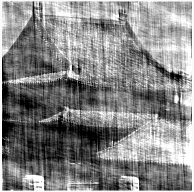
Modulus and phase of a digital image

Exchanging the modulus and the phase of two images:

Image 1

Image 2

Modulus of 1
& phase of 2



Modulus of 2
& phase of 1

- Geometric contours are mostly contained in the phase.

Modulus and phase of a digital image

Exchanging the modulus and the phase of two images:

Image 1

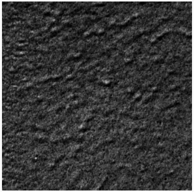
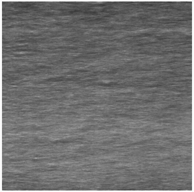


Image 2

Modulus of 1
& phase of 2



Modulus of 2
& phase of 1

- Textures are mostly contained in the modulus.

Modulus and phase of a digital image

Exchanging the modulus and the phase of two images:

Image 1

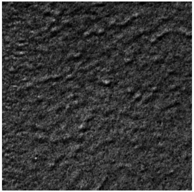
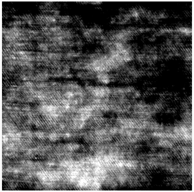
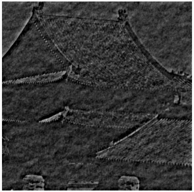


Image 2

Modulus of 1
& phase of 2

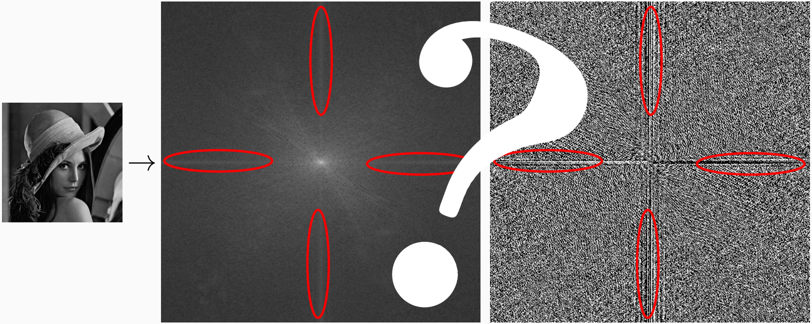




Modulus of 2
& phase of 1

- Geometric contours are mostly contained in the phase.
- Textures are mostly contained in the modulus.

Spectral filtering – 2d DFT



Amplitude   Phase

Why do the vertical and horizontal directions appear so strong?

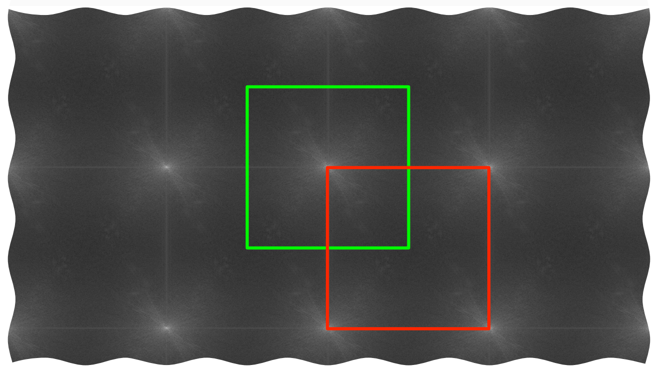
Spectral filtering – 2d DFT

$\longrightarrow$

Periodization

- It is assumed that the image is periodical
- Image borders may create strong edges
- Strong vertical and horizontal directions

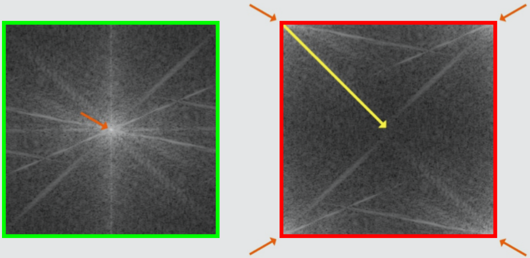
Spectral filtering – 2d DFT



Periodization

- The spectrum is also periodical
- Different ways to represent it

Recenter / Shift

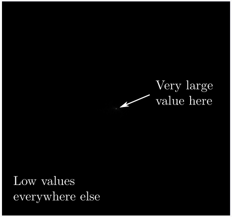


- Option 1: place the zero-frequency in the middle
  - Good way to visualize it
- Option 2: place the zero-frequency at top left location
  - Good way to manipulate it, used by `Python`, `Matlab`,...
- In `numpy`:
  - Forward transform: `dtfu = np.fft.fftshift(np.fft.fft2(u))`
  - Inverse transform: `v = np.fft.ifft2(np.fft.ifftshift(dtfu))`

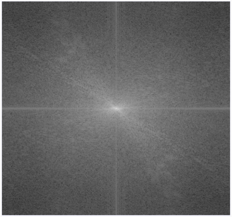
Visualization of the amplitude spectrum

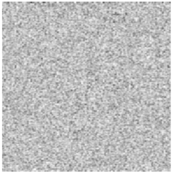
- Recall that $\hat{f}_{u,v} = \sum_{k=0}^{n_1-1} \sum_{l=0}^{n_2-1} f_{k,l} e^{-i2\pi\left(\frac{uk}{n_1} + \frac{vl}{n_2}\right)}$
- Then $\hat{f}_{0,0} = \sum_{k=0}^{n_1-1} \sum_{l=0}^{n_2-1} f_{k,l} = \sum$ of all intensities

  $\longleftarrow$ Can be very large!
- **Consequence:** the dynamic is too large to be displayed correctly
- **Solution:** perform a punctual non-linear transform
- **Classical one:** use $\log(|\hat{f}_{u,v}| + \varepsilon)$, $\varepsilon > 0$

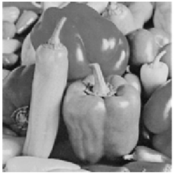


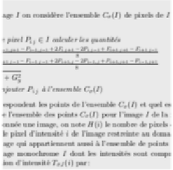
$\longrightarrow$

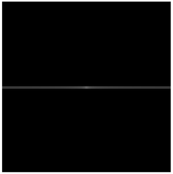


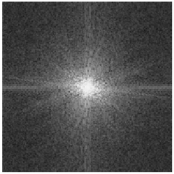

A

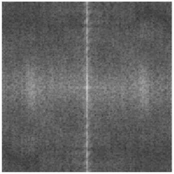

B

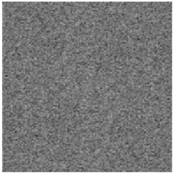

C


D


1


2


3


4

Which one is which?

Principle of spectral filtering

1. Apply the **Fourier transform**: $\hat{f} = \mathcal{F}[f]$
2. **Extract** the amplitude and phase

$$a_{u,v} = |\hat{f}_{u,v}| = \sqrt{\mathrm{Re}[\hat{f}_{u,v}]^2 + \mathrm{Im}[\hat{f}_{u,v}]^2}$$
$$\text{and} \quad \varphi_{u,v} = \arg \hat{f}_{u,v} = \mathrm{atan2}(\mathrm{Im}[\hat{f}_{u,v}], \mathrm{Re}[\hat{f}_{u,v}])$$

3. **Modify** the amplitude spectrum (and eventually the phase spectrum)

$$a_{u,v} \leftarrow a'_{u,v} \quad \text{and} \quad \varphi_{u,v} \leftarrow \varphi'_{u,v}$$

4. **Reconstruct** a complex spectrum

$$\hat{f}'_{u,v} = a'_{u,v} e^{i\varphi'_{u,v}}$$

5. Apply the **inverse Fourier transform**: $f' = \mathcal{F}^{-1}[\hat{f}']$

Useful only if we have a fast implementation of the Fourier transform

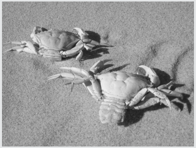
Discrete Fourier Transform (DFT)

$$\hat{f}_u = \sum_{k=0}^{n-1} f_k e^{-i2\pi \frac{uk}{n}} \quad \rightarrow \quad \text{Perform one loop for } u = 0 \text{ to } n-1$$

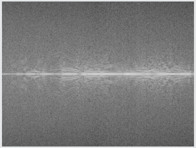
$$\rightarrow \quad \text{Direct computation in } O(n^2)$$

2d Discrete Fourier Transform (DFT2)

- The discrete Fourier transform is directionally separable



$\xrightarrow[\text{DFT}]{\text{Vertical}}$



$\xrightarrow[\text{DFT}]{\text{Horizontal}}$

- Complexity in: $O(n_1 n_2^2 + n_2 n_1^2) = O(n(n_1 + n_2))$
- Best scenario $n_1 = n_2 = \sqrt{n}$: $O(n^{3/2})$

Spectral filtering – Fast Fourier Transform

Fast Fourier Transform (FFT) [Cooley & Tukey, 1965]

- $\sim$1805: first described by Gauss (Fourier's paper: 1807)
- Exploits symmetry of DFT for faster computation
- Computation of the discrete Fourier transform can be done in

$$O(n \log n)$$

- Same for images thanks to directional separability

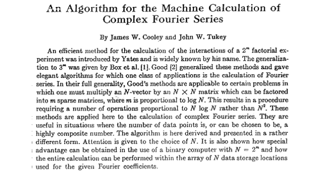
$$O(n_1 n_2 \log n_2 + n_2 n_1 \log n_1) = O(n(\log n_2 + \log n_1)) = O(n \log n)$$


Fourier


Gauss


John Tukey

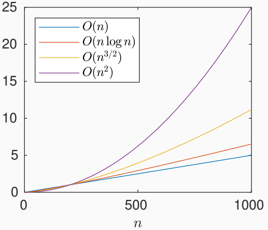

James Cooley


An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2. These methods are applied here to the calculation of complex Fourier series. They are useful in situations where the number of data points is, or can be chosen to be, a highly composite number. The algorithm is here derived and presented in a rather different form. Attention is given to the choice of N. It is also shown how special advantage can be obtained in the use of a binary computer with $N = 2^m$ and how the entire calculation can be performed within the array of N data storage locations used for the given Fourier coefficients.

J. W. Cooley and J. W. Tukey, Mathematics of Computation, Vol. 19, pp. 297-301, 1965.



(Source: Iasonas Kokkinos)

Spectral filtering – Fast Fourier Transform

FFT: Top 10 Algorithms of 20th Century!

Society for Industrial and Applied Mathematics (SIAM)
The Best of the 20th Century: Editors NameTop 10 Algorithms
May 16, 2000 Barry A Cipra

- 1946: The Metropolis Algorithm for Monte Carlo. Through the use of random processes, this algorithm offers an efficient way to stumble toward answers to problems that are too complicated to solve exactly.
- 1947: Simplex Method for Linear Programming. An elegant solution to a common problem in planning and decision-making.
- 1950: Krylov Subspace Iteration Method. A technique for rapidly solving the linear equations that abound in scientific computation.
- 1951: The Decompositional Approach to Matrix Computations. A suite of techniques for numerical linear algebra.
- 1957: The Fortran Optimizing Compiler. Turns high-level code into efficient computer-readable code.
- 1959: QR Algorithm for Computing Eigenvalues. Another crucial matrix operation made swift and practical.
- 1962: Quicksort Algorithms for Sorting. For the efficient handling of large databases.
- 1965: Fast Fourier Transform. Perhaps the most ubiquitous algorithm in use today, it breaks down waveforms (like sound) into periodic components.
- 1977: Integer Relation Detection. A fast method for spotting simple equations satisfied by collections of seemingly unrelated numbers.
- 1987: Fast Multipole Method. A breakthrough in dealing with the complexity of n-body calculations, applied in problems ranging from celestial mechanics to protein folding.

(Source: Ioannos Kokkinos)

Spectral filtering – Low-pass filter

Python demo – Low-pass filter

```
import numpy.fft as nf
import imagetools as im

▶ f      = plt.imread('butterfly.png')
n1, n2 = f.shape
tf     = nf.fft2(f, axes=(0, 1))
a      = np.abs(tf)
phi    = np.angle(tf)
u, v   = im.fftgrid(n1, n2)
dist2  = u**2 + v**2
mask   = dist2 <= r**2
ap     = mask * a
tfp    = ap * np.exp(1j * phi)
fp     = np.real(nf.ifft2(tfp, axes=(0, 1)))
```

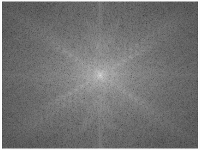
f

Spectral filtering – Low-pass filter

Python demo – Low-pass filter

```
import numpy.fft as nf
import imagetools as im

f      = plt.imread('butterfly.png')
n1, n2 = f.shape
tf     = nf.fft2(f, axes=(0, 1))
▶ a    = np.abs(tf)
phi    = np.angle(tf)
u, v   = im.fftgrid(n1, n2)
dist2  = u**2 + v**2
mask   = dist2 <= r**2
ap     = mask * a
tfp    = ap * np.exp(1j * phi)
fp     = np.real(nf.ifft2(tfp, axes=(0, 1)))
```

f



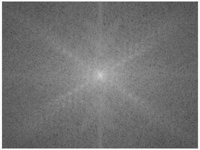
a

Spectral filtering – Low-pass filter

Python demo – Low-pass filter

```
import numpy.fft as nf
import imagetools as im

f      = plt.imread('butterfly.png')
n1, n2 = f.shape
tf     = nf.fft2(f, axes=(0, 1))
a      = np.abs(tf)
phi    = np.angle(tf)
▶ u, v = im.fftgrid(n1, n2)
dist2  = u**2 + v**2
mask   = dist2 <= r**2
ap     = mask * a
tfp    = ap * np.exp(1j * phi)
fp     = np.real(nf.ifft2(tfp, axes=(0, 1)))
```

f



a

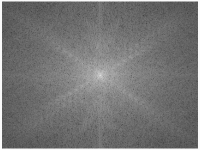
u

v

Python demo – Low-pass filter

```
import numpy.fft as nf
import imagetools as im

f      = plt.imread('butterfly.png')
n1, n2 = f.shape
tf     = nf.fft2(f, axes=(0, 1))
a      = np.abs(tf)
phi    = np.angle(tf)
u, v   = im.fftgrid(n1, n2)
▶ dist2 = u**2 + v**2
mask   = dist2 <= r**2
ap     = mask * a
tfp    = ap * np.exp(1j * phi)
fp     = np.real(nf.ifft2(tfp, axes=(0, 1)))
```

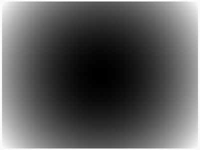
f



a

u

v



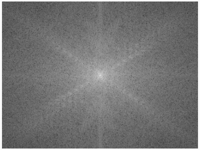
dist2

Spectral filtering – Low-pass filter

Python demo – Low-pass filter

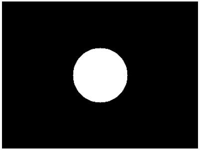
```
import numpy.fft as nf
import imagetools as im

f      = plt.imread('butterfly.png')
n1, n2 = f.shape
tf     = nf.fft2(f, axes=(0, 1))
a      = np.abs(tf)
phi    = np.angle(tf)
u, v   = im.fftgrid(n1, n2)
dist2  = u**2 + v**2
mask   = dist2 <= r**2
ap     = mask * a
tfp    = ap * np.exp(1j * phi)
fp     = np.real(nf.ifft2(tfp, axes=(0, 1)))
```


f


a


u


v

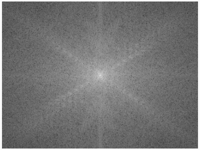

mask

Spectral filtering – Low-pass filter

Python demo – Low-pass filter

```
import numpy.fft as nf
import imagetools as im

f      = plt.imread('butterfly.png')
n1, n2 = f.shape
tf     = nf.fft2(f, axes=(0, 1))
a      = np.abs(tf)
phi    = np.angle(tf)
u, v   = im.fftgrid(n1, n2)
dist2  = u**2 + v**2
mask   = dist2 <= r**2
ap     = mask * a
tfp    = ap * np.exp(1j * phi)
fp     = np.real(nf.ifft2(tfp, axes=(0, 1)))
```

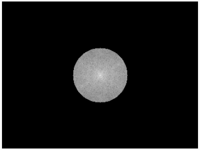
f



a

u

v



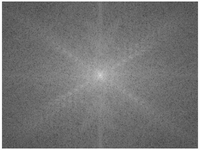
ap

Spectral filtering – Low-pass filter

Python demo – Low-pass filter

```
import numpy.fft as nf
import imagetools as im

f      = plt.imread('butterfly.png')
n1, n2 = f.shape
tf     = nf.fft2(f, axes=(0, 1))
a      = np.abs(tf)
phi    = np.angle(tf)
u, v   = im.fftgrid(n1, n2)
dist2  = u**2 + v**2
mask   = dist2 <= r**2
ap     = mask * a
tfp    = ap * np.exp(1j * phi)
▶ fp     = np.real(nf.ifft2(tfp, axes=(0, 1)))
```

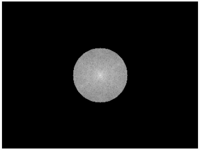
f



a

u

v



ap

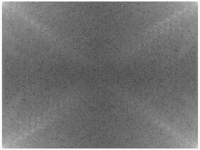
fp

Spectral filtering – Low-pass filter

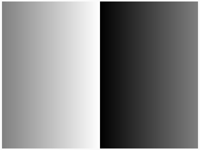
Python demo – Low-pass filter

```python
import numpy.fft as nf
import imagetools as im

f      = plt.imread('butterfly.png')
n1, n2 = f.shape
tf     = nf.fft2(f, axes=(0, 1))
a      = np.abs(tf)
phi    = np.angle(tf)
u, v   = im.fftgrid(n1, n2)
dist2  = u**2 + v**2
mask   = dist2 <= r**2
ap     = mask * a
tfp    = ap * np.exp(1j * phi)
fp     = np.real(nf.ifft2(tfp, axes=(0, 1)))
```

f



a



u

v

ap

fp

nf.fftshift

Spectral filtering – Low-pass filter

Shorter version

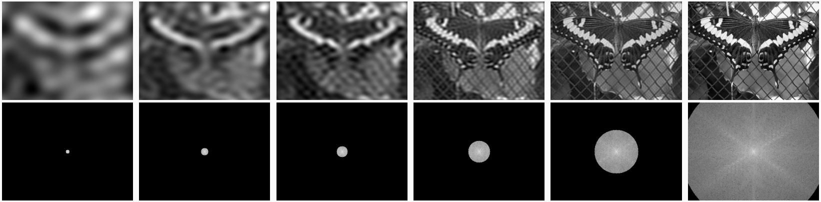
```
f      = plt.imread('butterfly.png')
n1, n2 = f.shape
u, v   = im.fftgrid(n1, n2)

tfp    = nf.fft2(f, axes=(0, 1))                # Transform
tfp[u**2 + v**2 > r**2] = 0                     # Modify
fp     = np.real(npf.ifft2(tfp, axes=(0, 1)))   # Transform back
```

Spectral filtering – Low-pass filter

Shorter version

```
f      = plt.imread('butterfly.png')
n1, n2 = f.shape
u, v   = im.fftgrid(n1, n2)

tfp    = nf.fft2(f, axes=(0, 1))                   # Transform
tfp[u**2 + v**2 > r**2] = 0                        # Modify
fp     = np.real(npf.ifft2(tfp, axes=(0, 1)))     # Transform back
```

What is the influence of the radius r?

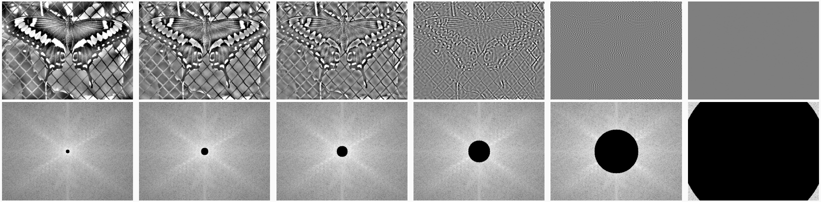


Acts similarly as a blur

Spectral filtering – High-pass filter

What if we do the opposite? (high-pass filter)

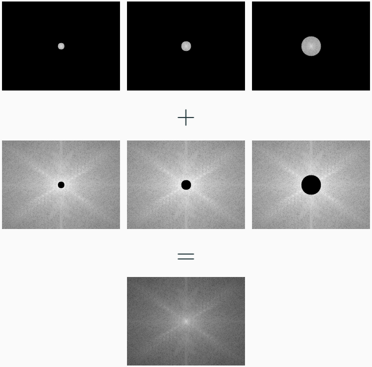
`u**2 + v**2 > r**2` $\rightarrow$ `u**2 + v**2 <= r**2`

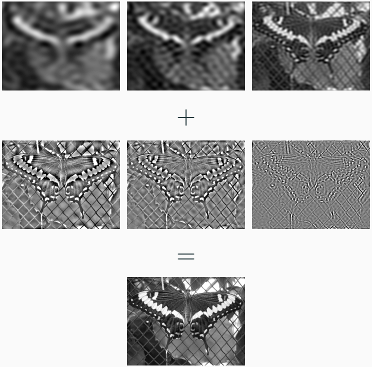


Acts similarly as an edge detector

Spectral filtering – High + Low -pass filters

What if we sum the two components?





$$\mathrm{M} \odot \hat{f} + (\mathrm{Id} - \mathrm{M}) \odot \hat{f} = \hat{f}$$

$$\mathcal{F}^{-1}[\mathrm{M} \odot \hat{f}] + \mathcal{F}^{-1}[(\mathrm{Id} - \mathrm{M}) \odot \hat{f}] = f$$

Image = Low frequencies + High frequencies
= Local averages + Edges/Textures

Spectral filtering – Low/High $\equiv$ Smooth/Edges

Standard spectral filters

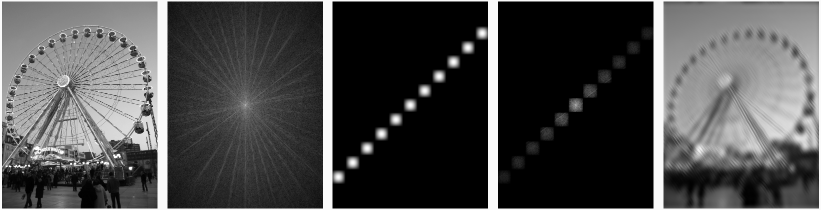
- Accept or reject some frequencies
- Low-pass filter: smooth the image (accept low frequencies)
- High-pass filter: preserve edges (accept high frequencies)

Is there a connection with moving averages and derivative filters?

Spectral filtering – Spectral modulation

Spectral modulation

- Apply the Fourier transform $\hat{x} = \mathcal{F}[x]$
- Modulate each frequency individually $\hat{y}_{u,v} = \lambda_{u,v} \cdot \hat{x}_{u,v}$
- Apply the inverse Fourier transform $y = \mathcal{F}^{-1}[\hat{y}]$



(a) x (b) $\hat{x}$ (c) λ (d) $\hat{y}$ (e) y

Spectral filtering – DFT in matrix form

$$\hat{x} = \mathcal{F}[x] \qquad \hat{y}_u = \lambda_u \cdot \hat{x}_u \qquad y = \mathcal{F}^{-1}[\hat{y}]$$

Matrix form in 1d

- The Fourier transform can be written as

$$\underbrace{\hat{x}_u = \sum_{k=0}^{n-1} x_k e^{-i2\pi \frac{uk}{n}}}_{=\mathcal{F}[x]_u} \quad \equiv \quad \hat{x} = \underbrace{\begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & e^{-i2\pi\frac{1}{n}} & \dots & e^{-i2\pi\frac{n-1}{n}} \\ 1 & e^{-i2\pi\frac{2}{n}} & \dots & e^{-i2\pi\frac{2(n-1)}{n}} \\ \vdots & & & \\ 1 & e^{-i2\pi\frac{(n-1)}{n}} & \dots & e^{-i2\pi\frac{(n-1)^2}{n}} \end{pmatrix}}_{=\boldsymbol{F}} x$$

- The modulation as: $\hat{y} = \underbrace{\begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix}}_{\boldsymbol{\Lambda}} \hat{x}$

- The inverse transform as $y = \boldsymbol{F}^{-1}\hat{y}$ with $\boldsymbol{F}^{-1} = \frac{1}{n}\boldsymbol{F}^*$.
- It follows that:

$$y = \frac{1}{n}\boldsymbol{F}^*\boldsymbol{\Lambda}\boldsymbol{F}x$$

Link with circulant matrices

- Let $\boldsymbol{E} = \frac{1}{\sqrt{n}}\boldsymbol{F}^*$ and $\boldsymbol{E}^{-1} = \frac{1}{\sqrt{n}}\boldsymbol{F}$, and write

$$y = \frac{1}{n}\boldsymbol{F}^*\boldsymbol{\Lambda}\boldsymbol{F}x = \boldsymbol{E}\boldsymbol{\Lambda}\boldsymbol{E}^{-1}x$$

- The columns of $\boldsymbol{E}$ are of the form

$$e_k = \frac{1}{\sqrt{n}}\left(1, \exp\left(\frac{2\pi ik}{n}\right), \exp\left(\frac{4\pi ik}{n}\right), \ldots, \exp\left(\frac{2(n-1)\pi ik}{n}\right)\right)^T$$

  and are eigenvectors with unit norms of circulant matrices (see, last class)
- Then $\boldsymbol{E}\boldsymbol{\Lambda}\boldsymbol{E}^{-1}$ is the eigendecomposition of a circulant matrix $\boldsymbol{H}$
- And $y = \boldsymbol{H}x$ is nothing else as the convolution of x by some kernel ν.

Convolutions are diagonal in the Fourier domain

Link with circulant matrices

- Let $\boldsymbol{E} = \frac{1}{\sqrt{n}}\boldsymbol{F}^*$ and $\boldsymbol{E}^{-1} = \frac{1}{\sqrt{n}}\boldsymbol{F}$, and write

$$y = \frac{1}{n}\boldsymbol{F}^*\boldsymbol{\Lambda}\boldsymbol{F}x = \boldsymbol{E}\boldsymbol{\Lambda}\boldsymbol{E}^{-1}x$$

- The columns of $\boldsymbol{E}$ are of the form

$$e_k = \frac{1}{\sqrt{n}}\left(1, \exp\left(\frac{2\pi ik}{n}\right), \exp\left(\frac{4\pi ik}{n}\right), \ldots, \exp\left(\frac{2(n-1)\pi ik}{n}\right)\right)^T$$

  and are eigenvectors with unit norms of circulant matrices (see, last class)
- Then $\boldsymbol{E}\boldsymbol{\Lambda}\boldsymbol{E}^{-1}$ is the eigendecomposition of a circulant matrix $\boldsymbol{H}$
- And $y = \boldsymbol{H}x$ is nothing else as the convolution of x by some kernel ν.

Convolutions are diagonal in the Fourier domain

Why is that important?

Spectral filtering – Fast convolutions with FFT

FFT $\Rightarrow$ Fast Convolutions

- Complexity of convolutions in spatial domain

- Limited support $s \times s$
  - Non separable: $O(s^2 n)$
  - Separable: $O(sn)$
- Unlimited support
  - Non separable: $O(n^2)$
  - Separable: $O(n^{3/2})$

- Complexity of convolutions through Fourier domain

$$\underbrace{\hat{x} = \mathcal{F}[x]}_{O(n \log n)} \qquad \underbrace{\hat{y}_u = \lambda_u \cdot \hat{x}_u}_{O(n)} \qquad \underbrace{y = \mathcal{F}^{-1}[\hat{y}]}_{O(n \log n)} \qquad \Rightarrow \quad \boldsymbol{O(n \log n)}$$

- Allows kernel functions to have a much larger support $s \times s$,
- Note: Spatial implementation can still be faster for small s.

Spectral filtering – Fast convolutions with FFT

FFT $\Rightarrow$ Fast Convolutions

- Complexity of convolutions in spatial domain
- Limited support $s \times s$
  - Non separable: $O(s^2 n)$
  - Separable: $O(sn)$
- Unlimited support
  - Non separable: $O(n^2)$
  - Separable: $O(n^{3/2})$
- Complexity of convolutions through Fourier domain

$$\underbrace{\hat{x} = \mathcal{F}[x]}_{O(n \log n)} \qquad \underbrace{\hat{y}_u = \lambda_u \cdot \hat{x}_u}_{O(n)} \qquad \underbrace{y = \mathcal{F}^{-1}[\hat{y}]}_{O(n \log n)} \qquad \Rightarrow \quad \boldsymbol{O(n \log n)}$$

- Allows kernel functions to have a much larger support $s \times s$,
- Note: Spatial implementation can still be faster for small s.

What is the link between the modulation λ and the convolution kernel ν?

Link between λ and ν

- The eigenvalues of a circulant matrix

$$\boldsymbol{H} = \begin{pmatrix} \nu_0 & \nu_{n-1} & \nu_{n-2} & \cdots & \nu_2 & \nu_1 \\ \nu_1 & \nu_0 & \nu_{n-1} & \nu_{n-2} & \cdots & \nu_2 \\ & & \ddots & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ \nu_{n-1} & \nu_{n-2} & \cdots & \nu_2 & \nu_1 & \nu_0 \end{pmatrix}$$

are

$$\lambda_u = \sum_{k=0}^{n-1} \nu_k \exp\left(-\frac{2\pi i u k}{n}\right)$$

Spectral filtering – Spectrum and convolution kernels

Link between λ and ν

- The eigenvalues of a circulant matrix

$$\boldsymbol{H} = \begin{pmatrix} \nu_0 & \nu_{n-1} & \nu_{n-2} & \dots & \nu_2 & \nu_1 \\ \nu_1 & \nu_0 & \nu_{n-1} & \nu_{n-2} & \dots & \nu_2 \\ & & \ddots & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ \nu_{n-1} & \nu_{n-2} & \dots & \nu_2 & \nu_1 & \nu_0 \end{pmatrix}$$

are

$$\lambda_u = \sum_{k=0}^{n-1} \nu_k \exp\left(-\frac{2\pi i u k}{n}\right) = \mathcal{F}[\nu]_u$$

- Which means: $\boldsymbol{H} = \boldsymbol{F}^{-1}\boldsymbol{\Lambda}\boldsymbol{F}$ with $\boldsymbol{\Lambda} = \mathrm{diag}(\boldsymbol{F}\nu)$, and thus

$$\nu * x = \boldsymbol{F}^{-1}\,\mathrm{diag}(\boldsymbol{F}\nu)\boldsymbol{F}x$$

This is the **Convolution theorem**

Spectral filtering – Spectrum and convolution kernels

Theorem (Convolution theorem)

Vector form

$$h = f * g \quad \Leftrightarrow \quad \hat{h}_u = \hat{f}_u \cdot \hat{g}_u$$

Function form

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

Matrix-vector form

$$f * g = \underbrace{\boldsymbol{F}^{-1} \operatorname{diag}(\boldsymbol{F} f) \boldsymbol{F}}_{\textit{circulant matrix}} g$$

Take home message

Convolution in spatial domain = Product in Fourier domain

Provides a new interpretation for LTI filters

- The convolution kernel ν characterizes the filter, (impulse response)
- Its Fourier transform $\lambda = \boldsymbol{F}\nu$ as well. (frequential response)

Spectral filtering – Properties of the Fourier transform

Main properties

	Time	Continuous	Discrete (periodic)
Linearity	$af + bg$	$a\hat{f} + b\hat{g}$	
Real/Hermitian	real	Hermitian	
Reverse/Conjugation	$f(-t)$	$\hat{f}^*$	
Convolution	$f * g$	$\hat{f} \cdot \hat{g}$	
Auto-correlation	$f \star g$	$\hat{f}^* \cdot \hat{g}$	
Zero frequency	$\int / \sum$	$\hat{f}(0)$	
Shift	$f(t-\delta)$	$e^{-i2\pi\delta u}\hat{f}(u)$	$e^{-i2\pi\delta u/n}\hat{f}_u$
Parseval	$\langle f, g \rangle$	$\langle \hat{f}, \hat{g} \rangle$	$\frac{1}{n}\langle \hat{f}, \hat{g} \rangle$
Plancherel	$\\|f\\|_2$	$\\|\hat{f}\\|_2$	$\frac{1}{n}\\|\hat{f}\\|_2$
Scaling	$f(at)$	$\frac{1}{\|a\|}\hat{f}(\frac{u}{a})$	–
Differentiation	$\frac{d^n f(t)}{dt^n}$	$(2\pi i u)^n \hat{f}(u)$	–

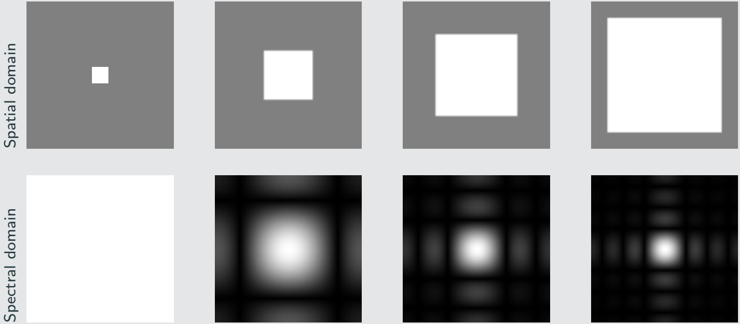
Similar properties for multi-dimensional signals

Properties of moving average filters

- Low frequencies are preserved
- High frequencies are attenuated
- Zero-frequency is always one
- Preserves the mean of pixel values

Spectral filtering – Moving averages = Low pass filters

Boxcar filter

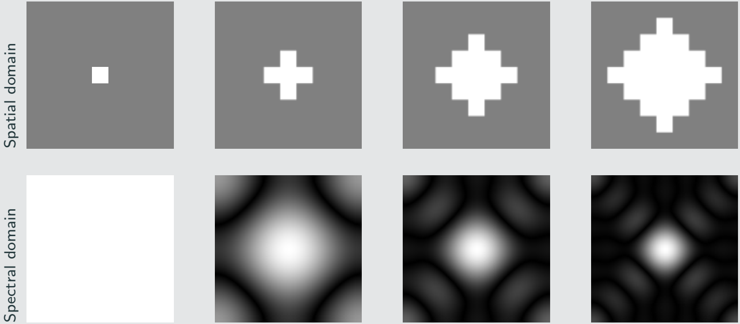


2d cardinal sines: $\frac{1}{\tau^2}\,\mathrm{sinc}(u/\tau)\,\mathrm{sinc}(v/\tau)$ $\qquad$ $\left(\mathrm{sinc}(t) = \dfrac{\sin(t)}{t}\right)$

- Bandwidth proportional to $1/\tau$
- Keep some high horizontal and vertical frequencies (side lobes)
- Explains horizontal and vertical artifacts of boxcar filters

Spectral filtering – Moving averages = Low pass filters

Diamond filter

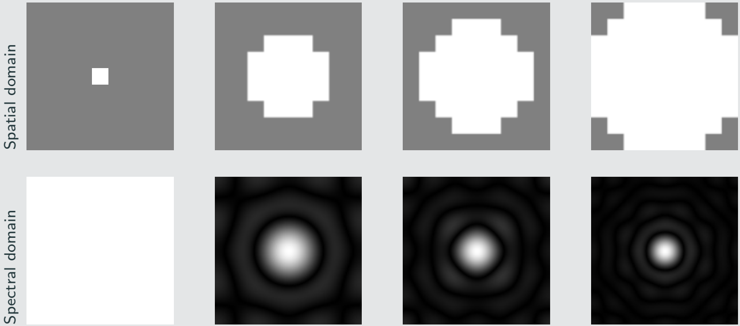


Similar to the box but rotated of 45°

- Bandwidth proportional to $1/\tau$
- Keep some high frequencies in diagonal directions (side lobes)
- Explains diagonal artifacts of diamond filters

Spectral filtering – Moving averages = Low pass filters

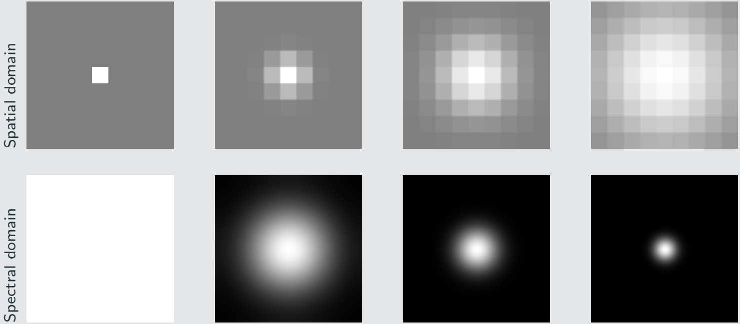
Diskcar filter



Cardinal sine in all directions

- Bandwidth proportional to $1/\tau$
- Keep some high frequencies (side lobes)
- No preferred direction (isotropic)

Gaussian filter

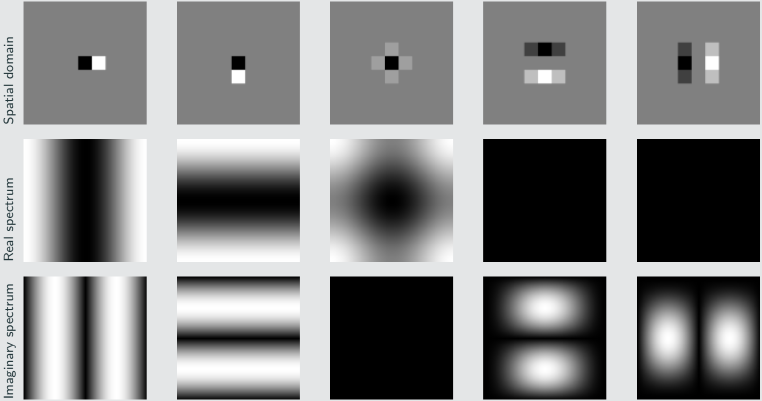


$$\mathcal{F}\left[\frac{1}{2\pi\tau^2} e^{-\frac{(s_1^2+s_2^2)}{2\tau^2}}\right] = e^{-4\pi^2\tau^2(u^2+v^2)} \qquad \equiv \qquad \mathcal{F}[\mathcal{G}_{\tau^2}] = \sqrt{2\pi\tau^2}^{d} \mathcal{G}_{1/4\pi^2\tau^2}$$

- Bandwidth proportional to $1/\tau$
- High frequencies are smoothly and monotonically removed
- No preferred direction (isotropic)

Spectral filtering – Derivative filters = High pass filters

Derivative filters = High pass filters

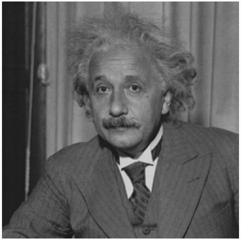


- Keep high frequencies only
- Often complex valued

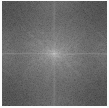
- Zero frequency is null
- Subtract the mean

Image sharpening

Spectral filtering – Image sharpening

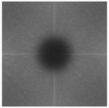


(a) x

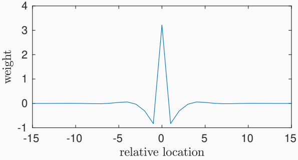


(b) $\hat{x}$

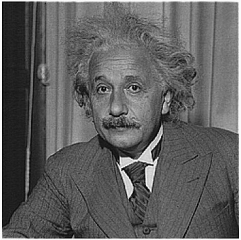
(c) $\lambda = \boldsymbol{F}\nu$



(d) $\hat{y}$



(e) ν



(f) y

Image sharpening

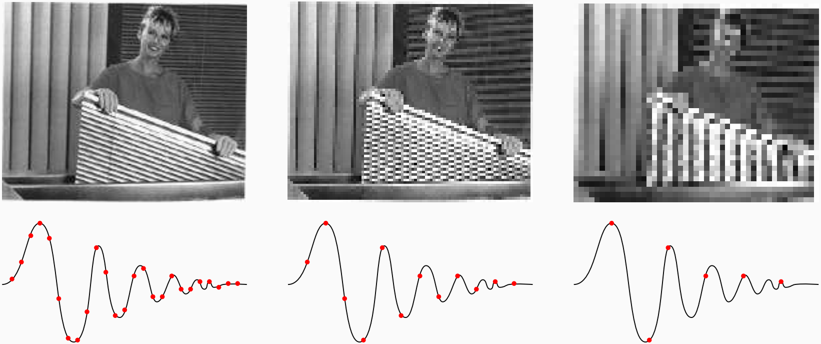
- Goal: Re-enforce edges
- How: $\left\{ \begin{array}{l} \bullet \text{ Keep low frequencies} \\ \bullet \text{ Amplify high frequencies} \end{array} \right.$
- Drawback: Amplify noise

$$y = x + \alpha D x$$

D: derivative filter, $\alpha > 0$

Image resizing and aliasing

Spectral filtering – Image resizing / sub-sampling



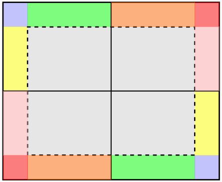
(a) $\times 1$ (b) $\times 2$ (c) $\times 4$

Spatial image resizing (sub-sampling by a factor a)

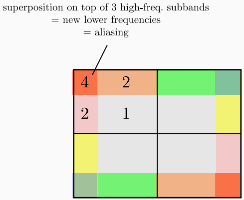
- Continuous image: $f^{\text{rescaled}}(t) = f(at)$
- Discrete image, ex: $f_k^{\text{rescaled}} = (1 - ak + \lfloor ak \rfloor) f_{\lfloor ak \rfloor} + (ak - \lfloor ak \rfloor) f_{\lceil ak \rceil}$ (linear interpolation)
- Aliasing: High frequencies lost, new frequencies created. Why?

Aliasing example

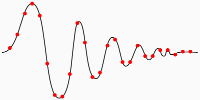
Spectral filtering – Image resizing / sub-sampling



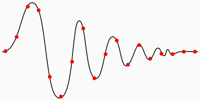
Spectrum before spatial subsampling



after spatial subsampling



(a) $\times 1$



(b) $\times 4/3$

Nyquist

Shannon

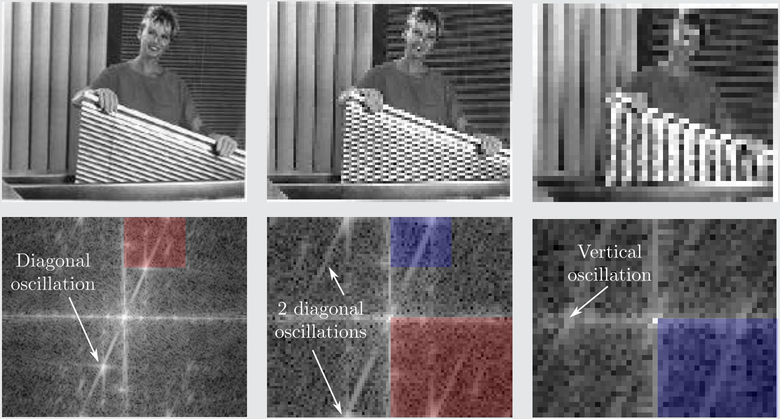
Aliasing

- Superposition of high frequency sub-bands in the new resized image
- Linked with Nyquist-Shannon's theorem:

  sampling frequency should be at least double the maximum frequency

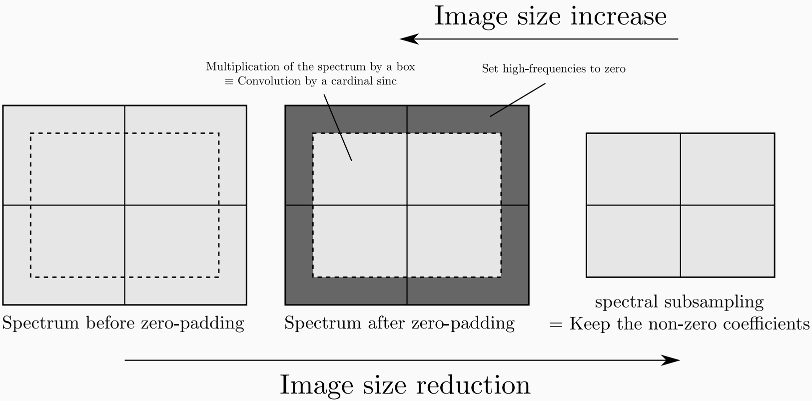
Spectral filtering – Image resizing / sub-sampling

Aliasing: how diagonal stripes become vertical...



How to avoid aliasing when resizing?

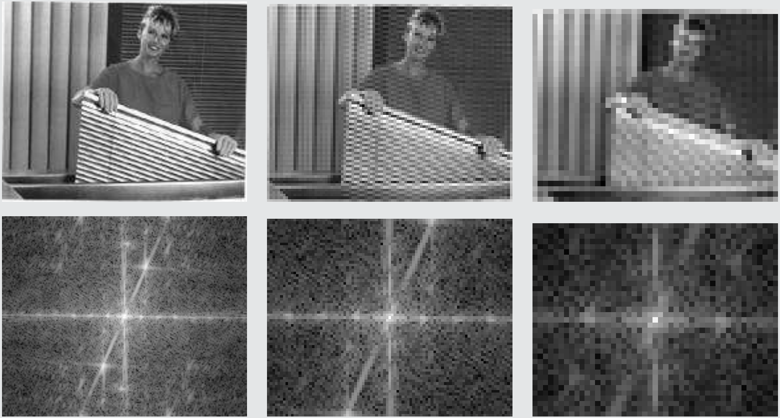
Spectral filtering – Image resizing / sub-sampling



Spectral image resizing with zero-padding

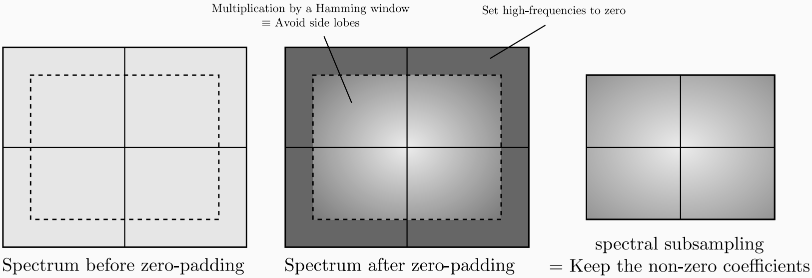
- Reduction: set high frequencies to zero and reduce spectrum size
- Increase: increase spectrum size and fill new high frequencies by zeros

Zero-padding: No more aliasing but unpleasant oscillations



How to avoid side lobes of the cardinal sine? (ringing/Gibbs artifacts)

Spectral filtering – Image resizing / sub-sampling



Zero-padding + windowing

- Not only set the high frequencies to zeros
- But modulate low frequencies by a weighting window, *i.e.*, a blur
- Choice of the window: trade-off between ringing vs blur

Typical windows

- Hann window: to reduce all side lobes

$$w(u) = 0.5 - 0.5\cos\left(\frac{2\pi(u + \lceil n/2\rceil - 1)}{n-1}\right)$$

- Hamming window: to reduce first side lobe

$$w(u) = 0.54 - 0.46\cos\left(\frac{2\pi(u + \lceil n/2\rceil - 1)}{n-1}\right)$$

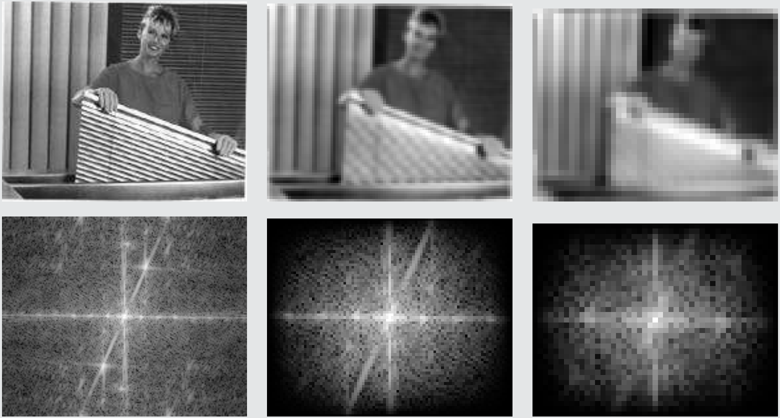
- Kaiser window: to choose a trade-off between blur and side lobes.

$$w(u) = \frac{I_0\left(\pi\alpha\sqrt{1 - \left(\frac{2(u+\lceil n/2\rceil - 1)}{n-1} - 1\right)^2}\right)}{I_0(\pi\alpha)}, \quad \alpha > 0$$

for frequencies $u = -\lceil n/2\rceil + 1$ to $\lfloor n/2\rfloor$.

I_0: zero-order modified Bessel function.

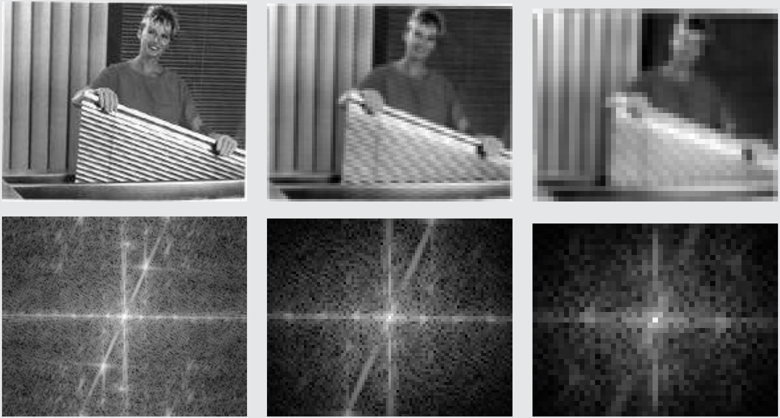
Hann window



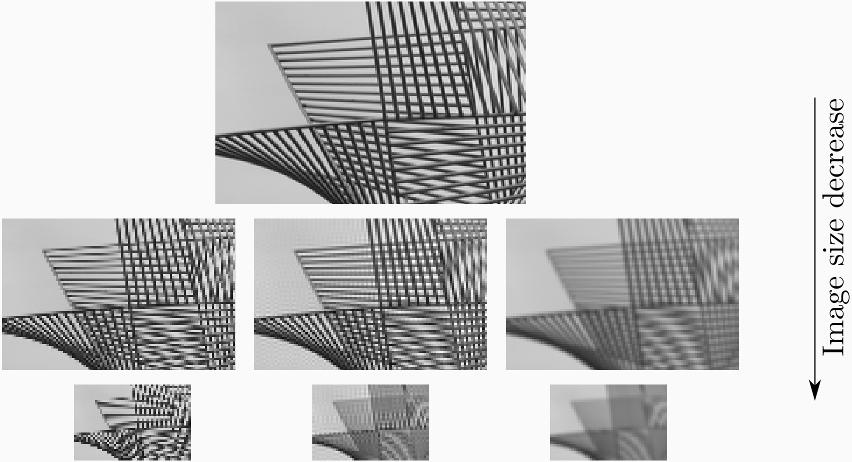
Hann window: No more aliasing, no more ringing, but blur

Spectral filtering – Image resizing / sub-sampling

Kaiser window



Kaiser window: No more aliasing and trade-off between ringing and blur

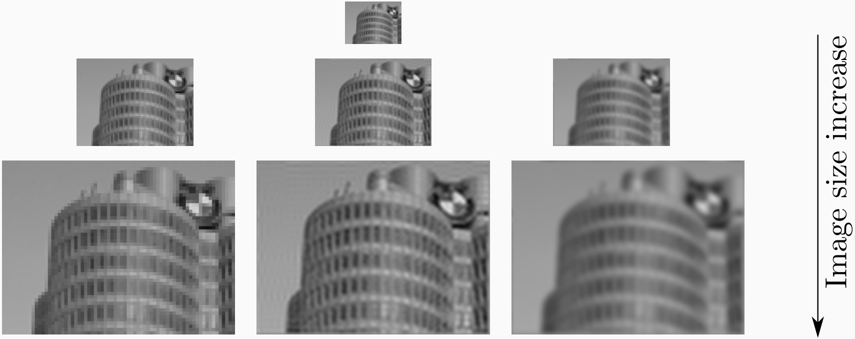


Spatial sub-sampling
Linear interpolation

Spectral sub-sampling
Zero-padding

Spectral sub-sampling
Windowing

Spectral filtering – Image resizing / sub-sampling



Spatial sub-sampling
Linear interpolation

Spectral sub-sampling
Zero-padding

Spectral sub-sampling
Windowing

Introduction to variational methods for inverse problems in image processing

Major image restoration issues

Usual image degradation models

- Images often viewed through a linear operator (*e.g.* blur)

$$y = \boldsymbol{H}x \quad \Leftrightarrow \quad \begin{cases} h_{11}x_1 + h_{12}x_2 + \ldots + h_{1n}x_n & = y_1 \\ h_{21}x_1 + h_{22}x_2 + \ldots + h_{2n}x_n & = y_2 \\ \vdots & \\ h_{n1}x_1 + h_{n2}x_2 + \ldots + h_{nn}x_n & = y_n \end{cases}$$

- Retrieving $x \Rightarrow$ Inverting $\boldsymbol{H}$ (*i.e.*, solving the system of linear equations)

$$\hat{x} = \boldsymbol{H}^{-1}y$$

(a) Unknown image x

$\xrightarrow{\boldsymbol{H}}$

(b) Observation y

$\xrightarrow{\boldsymbol{H}^{-1}}$

(c) Estimate $\hat{x}$

Is image restoration solved then?

Major image restoration issues

Limitations

- $\boldsymbol{H}$ is often non-invertible
  - equations are linearly dependent,
  - system is under-determined,
  - infinite number of solutions,
  - which one to choose?
- The system is said to be ill-posed in opposition to well-posed.

Well-posed problem **(Hadamard)**

1. a solution exists,
2. the solution is unique,
3. the solution's behavior changes continuously with the initial conditions.

Major image restoration issues

Limitations

- Or, $\boldsymbol{H}$ is invertible but ill-conditioned:
  - small perturbations in y lead to large errors in $\hat{x} = \boldsymbol{H}^{-1}y$,
  - and unfortunately y is often corrupted by noise: $y = \boldsymbol{H}x + w$,
  - and unfortunately y is often encoded with limited precision.

(a) Unknown image x (b) Observation y (c) Estimate $\hat{x}$

- Condition-number: $\kappa(\boldsymbol{H}) = \|\boldsymbol{H}^{-1}\|_2 \|\boldsymbol{H}\|_2 = \frac{\sigma_{\max}}{\sigma_{\min}}$

(σ_k singular values of $\boldsymbol{H}$)

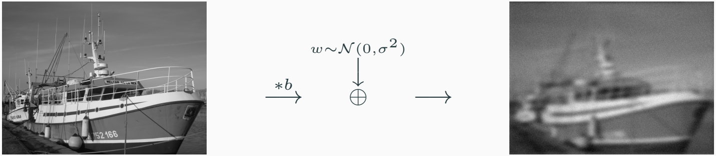
- the larger $\kappa(\boldsymbol{H}) \geqslant 1$, the more ill-conditioned/difficult is the inversion.

Typical problem setting: Non-blind image deblurring

- We consider the following degradation process: An image x is blurred by a kernel h (e.g. motion blur or out of focus) and additional white (Gaussian) noise with mean 0 and variance σ^2 is added.

$$y = b * x + n$$

- We suppose that both the blur kernel b and the noise variance σ^2 are known.
- If these important quantities are not know, we speak of blind deconvolution.



(a) Unknown image x (b) Observation y

Least square – Least square and normal equation

Pseudo-inverse approach: Find an image $\hat{x}$ solution of the optimization problem:

$$\min_x \|\boldsymbol{H}x - y\|_2^2$$

Least-square estimator and normal equation

- If H is not invertible, there are **infinite solutions** of the least square problem

$$\min_x \|\boldsymbol{H}x - y\|_2^2$$

- They are characterized by the **normal equation**

$$\boldsymbol{H}^*\boldsymbol{H}x^\star = \boldsymbol{H}^*y$$

- If initialized to zero, a gradient descent gives the one with minimum norm $\|\hat{x}^\star\|_2$.
- This solution reads as $$\hat{x}^\star = \boldsymbol{H}^+y$$

  where $\boldsymbol{H}^+ \in \mathbb{R}^{p\times n}$ is the Moore-Penrose pseudo-inverse of $\boldsymbol{H}$.

Moore-Penrose pseudo-inverse

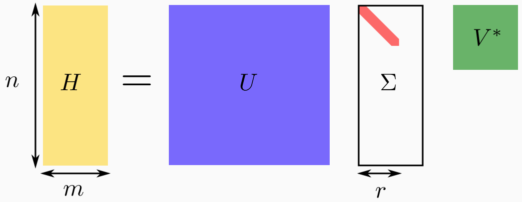
- The Moore-Penrose pseudo-inverse is the unique matrix satisfying
  1. $\boldsymbol{H}\boldsymbol{H}^{+}\boldsymbol{H} = \boldsymbol{H}$
  2. $\boldsymbol{H}^{+}\boldsymbol{H}\boldsymbol{H}^{+} = \boldsymbol{H}^{+}$
  3. $(\boldsymbol{H}\boldsymbol{H}^{+})^{*} = \boldsymbol{H}\boldsymbol{H}^{+}$
  4. $(\boldsymbol{H}^{+}\boldsymbol{H})^{*} = \boldsymbol{H}^{+}\boldsymbol{H}$
- The Moore-Penrose pseudo-inverse always exists.
- If $\boldsymbol{H}$ is square and invertible: $\boldsymbol{H}^{+} = \boldsymbol{H}^{-1}$
- $\boldsymbol{H}^{+}$ also satisfy: $\boldsymbol{H}^{+} = (\boldsymbol{H}^{*}\boldsymbol{H})^{+}\boldsymbol{H}^{*} = \boldsymbol{H}^{*}(\boldsymbol{H}\boldsymbol{H}^{*})^{+}$
- If $\boldsymbol{H}$ has full rank: $\boldsymbol{H}^{+} = (\boldsymbol{H}^{*}\boldsymbol{H})^{-1}\boldsymbol{H}^{*}$.

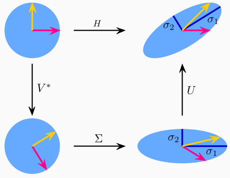
Small detour to Singular Value Decompositions (SVD)

- Any matrix $\boldsymbol{H} \in \mathbb{R}^{n \times m}$ admits a Singular Value Decomposition (SVD) as

$$\boldsymbol{H} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^* \quad \text{with} \quad \begin{cases} \bullet\ \boldsymbol{U} \in \mathbb{C}^{n \times n},\ \boldsymbol{U}^*\boldsymbol{U} = \boldsymbol{U}^*\boldsymbol{U} = \mathrm{Id}_n \\ \bullet\ \boldsymbol{V} \in \mathbb{C}^{m \times m},\ \boldsymbol{V}^*\boldsymbol{V} = \boldsymbol{V}\boldsymbol{V}^* = \mathrm{Id}_m \\ \bullet\ \boldsymbol{\Sigma} \in \mathbb{R}^{n \times m} \text{ a diagonal matrix.} \end{cases}$$

- $\sigma_i = \boldsymbol{\Sigma}_{ii} > 0$: called singular values (often sorted in decreasing order),
- Rank $r \leqslant \min(n, m)$: number of non-zero singular values.





$H = U \cdot \Sigma \cdot V^*$

Least square – Moore-Penrose pseudo-inverse

SVD, image and null space

- If the singular values are sorted in decreasing order

$$\begin{aligned}\mathrm{Im}[\boldsymbol{H}] &= \{y \in \mathbb{R}^n \;;\; \exists x \in \mathbb{R}^m,\; y = \boldsymbol{H}x\}\\ &= \mathrm{Span}(\{u_i \in \mathbb{R}^n \;;\; i \in [1 \ldots r]\}) \qquad \text{(what can be observed)}\end{aligned}$$

$$\begin{aligned}\mathrm{Ker}[\boldsymbol{H}] &= \{x \in \mathbb{R}^m \;;\; \boldsymbol{H}x = 0\}\\ &= \mathrm{Span}(\{v_i \in \mathbb{R}^m \;;\; i \in [r+1 \ldots m]\}) \qquad \text{(what is lost)}\end{aligned}$$

where u_i are the columns of $\boldsymbol{U}$ and v_i are the columns of $\boldsymbol{V}$

Null-space: set of zero-frequencies, set of missing pixels, ...

Least square – Moore-Penrose pseudo-inverse

SVD and Moore-Penrose pseudo-inverse

- Let $\boldsymbol{H} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$ be its SVD, the Moore-Penrose pseudo inverse is

$$\boldsymbol{H}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^* \quad \text{where} \quad \sigma_i^+ = \begin{cases} \frac{1}{\sigma_i} & \text{if } \sigma_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

- For deconvolution: SVD $\cong$ eigendecomposition $\cong$ Fourier decomposition
  $\Rightarrow$ inversion of the non-zero frequencies.

- Difficulty: $\frac{1}{\sigma_i}$ can be very large (ill-conditioned matrix)
  $\Rightarrow$ numerical issues.

- The space of images can be decomposed into the orthogonal sum:

$$\mathbb{R}^m = \operatorname{Ker}[\boldsymbol{H}] \overset{\perp}{\oplus} \operatorname{Im}[\boldsymbol{H}^T]$$

- For $f(x) = \|\boldsymbol{H}x - y\|_2^2$,

$$\nabla f(x) = 2\boldsymbol{H}^T(\boldsymbol{H}x - y) \in \operatorname{Im}[\boldsymbol{H}^T].$$

- Compared to other least square solutions, the Moore-Penrose pseudo-inverse does not create new content in $\operatorname{Ker}[\boldsymbol{H}]$.
- More generally, using only f for gradient descent, one keeps the $\operatorname{Ker}[\boldsymbol{H}]$ component of the initialization (0 for pseudo-inverse).
- **Solution:** Change the function to minimize by incorporating a priori on "good" images

Variational methods

Definition

A variational problem is as an **optimization problem** of the form

$$\min_x \left\{ F(x) = \int_\Omega f(s, x, \nabla x) \, \mathrm{d}s \right\}$$

where

- Ω: image support (ex: $[0,1]^2$),
- $x : \Omega \mapsto \mathbb{R}$: function that maps a position s to a value,
- $\nabla x : \Omega \mapsto \mathbb{R}^2$: gradient of x,
- $s = (s_1, s_2) \in \Omega$: space location,
- $f(s, p, v)$: loss chosen for a given task,
- F: functional that maps a function to a value. (function of a function)

Example (Tikhonov functional)

- Consider the inverse problem $y = H(x) + w$, with H linear.
- The Tikhonov functional F is, for $\tau > 0$, defined as

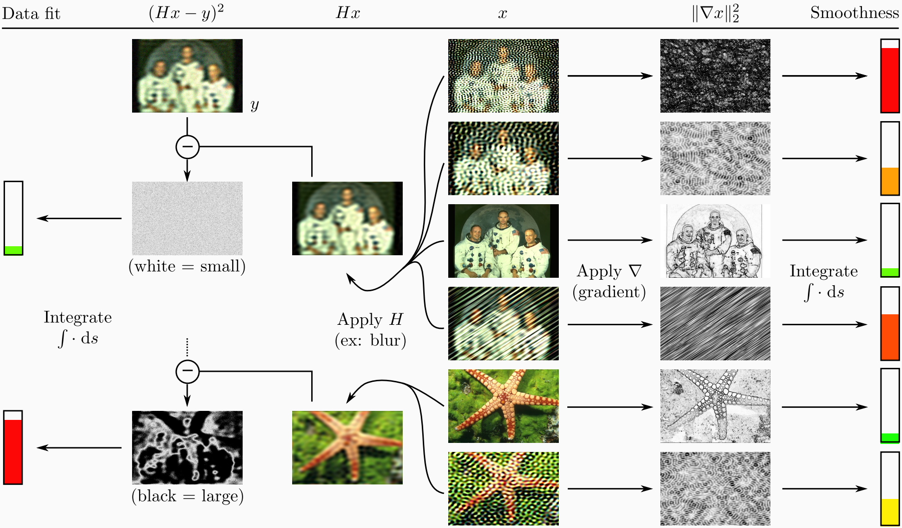
$$F(x) = \frac{1}{2} \int_\Omega (H(x)(s) - y(s))^2 + \tau \|\nabla x(s)\|_2^2 \, \mathrm{d}s$$

or, in short, we write

$$= \frac{1}{2} \int_\Omega \underbrace{(H(x) - y)^2}_{\text{data fit}} + \tau \underbrace{\|\nabla x\|_2^2}_{\text{smoothing}} \, \mathrm{d}s$$

- Look for x such that its degraded version $H(x)$ is close to y.
- But, discourage x to have large spatial variations.
- τ: regularization parameter (trade-off).

Variational methods - Tikhonov functional



Pick the image x with smallest: Data-fit + Smoothness

$$F(x) = \frac{1}{2} \int_\Omega \underbrace{(H(x) - y)^2}_{\text{data fit}} + \tau \underbrace{\|\nabla x\|_2^2}_{\text{smoothing}} \ \mathrm{d}s$$

Example (Tikhonov functional)

- The image x is forced to be close to the noisy image y through H, but the amplitudes of its gradient are penalized to avoid overfitting the noise.
- The parameter $\tau > 0$ controls the regularization.
- For $\tau \to 0$, the problem becomes ill-posed/ill-conditioned, noise remains and may be amplified.
- For $\tau \to \infty$, x tends to be constant (depends on boundary conditions).

(a) Low resolution y

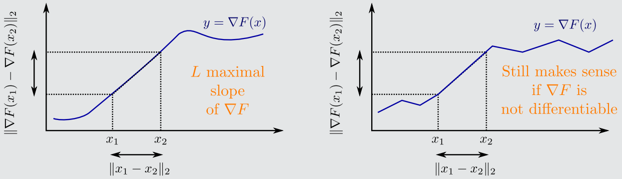
Tikhonov regularization for $\times$ 16 super-resolution

(b) $\tau = 0$ (c) Small τ (d) Good τ (e) High τ (f) $\tau \to \infty$

Lipschitz gradient

- A differentiable function F has L Lipschitz gradient, if

$$\|\nabla F(x_1) - \nabla F(x_2)\|_2 \leqslant L\|x_1 - x_2\|_2, \quad \text{for all } x_1,\ x_2\ .$$



- The mapping $x \mapsto \nabla F(x)$ is necessarily continuous.
- If F is twice differentiable

$$L = \sup_x \ \| \underbrace{\nabla^2 F(x)}_{\text{Hessian matrix of } F} \|_2.$$

where for a matrix A, its ℓ_2-norm $\|A\|_2$ is its maximal singular value.

Be careful:

- $\nabla x \in \mathbb{R}^{n \times 2}$ is a 2d discrete vector field, corresponding to the discrete gradient of the image x.

- $(\nabla x)_k \in \mathbb{R}^2$ is a 2d vector: the discrete gradient of x at location s_k.

- $\nabla F(x) \in \mathbb{R}^n$ is the (continuous) gradient of F at x.

- $(\nabla F(x))_k \in \mathbb{R}$: variation of F for an infinitessimal variation of the pixel value x_k.

Gradient descent

- Let F be a real function, differentiable and coercive with a L Lipschitz gradient. Then, whatever the initialization x^0, if $0 < \gamma < 2/L$, the sequence

$$x^{k+1} = x^k - \gamma \nabla F(x^k) \ ,$$

converges to a stationary point $x^\star$ (*i.e.*, it cancels the gradient)

$$\nabla F(x^\star) = 0 \ .$$

- The parameter γ is called the step size.
- A too small step size γ leads to slow convergence.
- For $0 < \gamma < 2/L$, the sequence $F(x^k)$ decays with a rate in $O(1/k)$.

Gradient descent for convex function

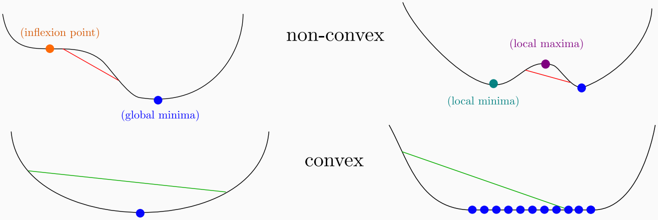
- If moreover F is convex

$$F(\lambda x_1 + (1-\lambda)x_2) \leqslant \lambda F(x_1) + (1-\lambda)F(x_2), \quad \forall x_1, x_2, \lambda \in (0,1) \ ,$$

then, the gradient descent converges towards a global minimum

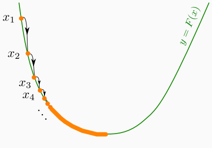
$$x^\star \in \underset{x}{\operatorname{argmin}}\ F(x).$$

- Note: All stationary points are global minimum (non necessarily unique).

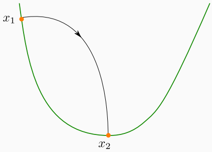


Variational methods – Smooth optimization

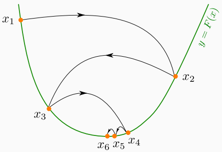
One-dimension



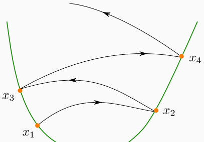
Small step size
Slow convergence



Good step size
Fast convergence

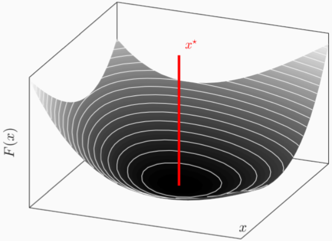


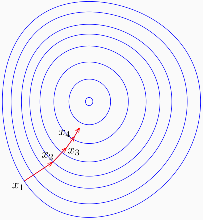
Large step size
Slow convergence

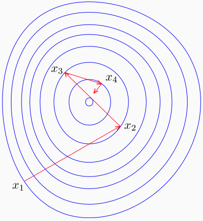


Too large step size
Divergence

Two-dimensions







Example (Tikhonov functional (1/6))

- The functional F is

$$F(x) = \frac{1}{2} \int_\Omega \underbrace{(H(x) - y)^2}_{\text{data fit}} + \tau \underbrace{\|\nabla x\|_2^2}_{\text{smoothing}} \; \mathrm{d}s \; .$$

- Its discretization leads to

$$\begin{aligned} F(x) &= \frac{1}{2} \sum_k ((\boldsymbol{H} x)_k - y_k)^2 + \frac{\tau}{2} \sum_k \|(\nabla x)_k\|_2^2 \\ &= \frac{1}{2} \|\boldsymbol{H} x - y\|_2^2 + \frac{\tau}{2} \|\nabla x\|_{2,2}^2 \end{aligned}$$

- Squared $\ell_{2,2}$/Frobenius norm of a matrix = sum of all coefficient to the square

$$\|\boldsymbol{A}\|_{2,2}^2 = \sum_k \|\boldsymbol{A}_k\|_2^2 = \sum_k \sum_l \boldsymbol{A}_{kl}^2 = \operatorname{tr} \boldsymbol{A}^* \boldsymbol{A} = \langle \boldsymbol{A}, \, \boldsymbol{A} \rangle \; .$$

- Scalar product between matrices: $\operatorname{tr} \boldsymbol{A}^* \boldsymbol{B} = \langle \boldsymbol{A}, \, \boldsymbol{B} \rangle$.

$$F(x) = \frac{1}{2}\|\boldsymbol{H}x - y\|_2^2 + \frac{\tau}{2}\|\nabla x\|_{2,2}^2$$

Example (Tikhonov functional (2/6))

- This function is differentiable and convex, since
  - If f convex, $x \mapsto f(\boldsymbol{A}x + b)$ is convex,
  - Norms are convex,
  - Quadratic functions are convex,
  - Compositions of convex non-decreasing functions (left) and convex functions (right) are convex.
  - Sums of convex functions are convex.
- We can solve this problem using gradient descent.

$$F(x) = \frac{1}{2}\|\boldsymbol{H}x - y\|_2^2 + \frac{\tau}{2}\|\nabla x\|_{2,2}^2$$

Example (Tikhonov functional (3/6))

- Note that $\|\nabla x\|_{2,2}^2 = \langle \nabla x, \, \nabla x \rangle = \langle x, \, -\operatorname{div} \nabla x \rangle = -\langle x, \, \Delta x \rangle$, then

$$\begin{aligned} F(x) &= \frac{1}{2}(\|\boldsymbol{H}x\|^2 + \|y\|^2 - 2\langle \boldsymbol{H}x, \, y \rangle) - \frac{\tau}{2}\langle x, \, \Delta x \rangle \\ &= \frac{1}{2}(\langle x, \, \boldsymbol{H}^*\boldsymbol{H}x \rangle + \|y\|^2 - 2\langle x, \, \boldsymbol{H}^* y \rangle) - \frac{\tau}{2}\langle x, \, \Delta x \rangle \end{aligned}$$

- The gradient is thus given by

$$\begin{aligned} \nabla F(x) &= \frac{1}{2}((\boldsymbol{H}^*\boldsymbol{H} + \boldsymbol{H}^*\boldsymbol{H})x - 2\boldsymbol{H}^* y - \tau(\Delta + \Delta^*)x) \\ &= \boldsymbol{H}^*(\boldsymbol{H}x - y) - \tau \Delta x \end{aligned}$$

Note: $\nabla \langle x, \, \boldsymbol{A}y \rangle = \boldsymbol{A}y$ and $\nabla \langle x, \, \boldsymbol{A}x \rangle = (\boldsymbol{A} + \boldsymbol{A}^*)x$

Example (Tikhonov functional (4/6))

- The gradient descent reads as

$$\begin{aligned} x^{k+1} &= x^k - \gamma \nabla F(x^k) \\ &= x^k - \gamma(\boldsymbol{H}^*(\boldsymbol{H}x^k - y) - \tau \Delta x^k) \end{aligned}$$

  with $\gamma < \frac{2}{L}$ where $L = \|\boldsymbol{H}^*\boldsymbol{H} - \tau\Delta\|_2$.

- Triangle inequality: $L \leqslant \|\boldsymbol{H}\|_2^2 + \tau 4d$ since $\|\Delta\|_2 = 4d$.

Example (Tikhonov functional (5/6))

$$x^{k+1} = x^k - \gamma(\underbrace{\boldsymbol{H}^*(\boldsymbol{H}x^k - y)}_{\text{retroaction}} - \tau\Delta x^k)$$

- The retroaction allows to remain close to the observation.
- Unlike the solution of the Heat equation,
  this numerical scheme converges to a solution of interest.
- Classical stopping criteria:
  - fixed number m of iterations ($k = 1$ to m),
  - $|F(x^{k+1}) - F(x^k)|/|F(x^k)| < \varepsilon$, or
  - $\|x^{k+1} - x^k\|/\|x^k\| < \varepsilon$.

Where does Tikhonov regularization converge to?

Example (Tikhonov regularization (6/6))

- Explicit solution

$$\nabla F(x) = \boldsymbol{H}^*(\boldsymbol{H}x - y) - \tau \Delta x = 0$$
$$\Leftrightarrow$$
$$x^\star = (\boldsymbol{H}^*\boldsymbol{H} - \tau\Delta)^{-1}\boldsymbol{H}^* y$$

- Can be directly solved by conjugate gradient.
- Tikhonov regularization is linear (non-adaptive).
- If $\boldsymbol{H}$ is a blur, this is a convolution by a sharpening kernel (LTI filter).

Variational methods - Tikhonov functional

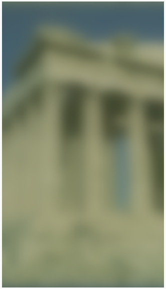
(a) Low resolution y

Tikhonov regularization for $\times$ 16 super-resolution

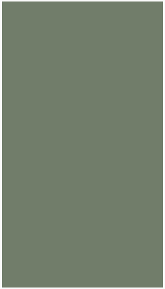
(b) $\tau = 0$

(c) Small τ

(d) Good τ



(e) High τ



(f) $\tau \to \infty$

Total-Variation

$$F(x) = \frac{1}{2} \int (\boldsymbol{H}x - y)^2 + \tau \|\nabla x\|_2 \, \mathrm{d}s$$

2d Total-Variation

- Its discretization leads to

$$\begin{aligned} F(x) &= \frac{1}{2}\|\boldsymbol{H}x - y\|_2^2 + \frac{\tau}{2} \sum_k \|(\nabla x)_k\|_2 \\ &= \frac{1}{2}\|\boldsymbol{H}x - y\|_2^2 + \frac{\tau}{2}\|\nabla x\|_{2,1} \end{aligned}$$

- $\|\nabla x\|_2$ is not differentiable at 0 values.
- Favors piecewise constant images.
- Need for more involved optimization algorithms.

Total-Variation – Results

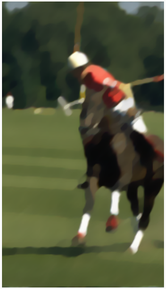
(a) Blurry image y

TV regularization for deconvolution of motion blur

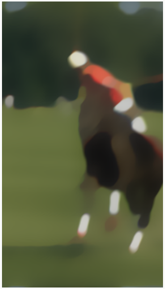
(b) Tiny τ

(c) Small τ

(d) Medium τ



(e) High τ



(f) Huge τ

Total-Variation – Results

(a) Blurry image y

Total-Variation – Results

(b) Tiny τ

(c) Small τ

Total-Variation – Results

(d) Relatively small τ

Total-Variation – Results

(e) Medium τ

(f) Large τ

Total-Variation – Results

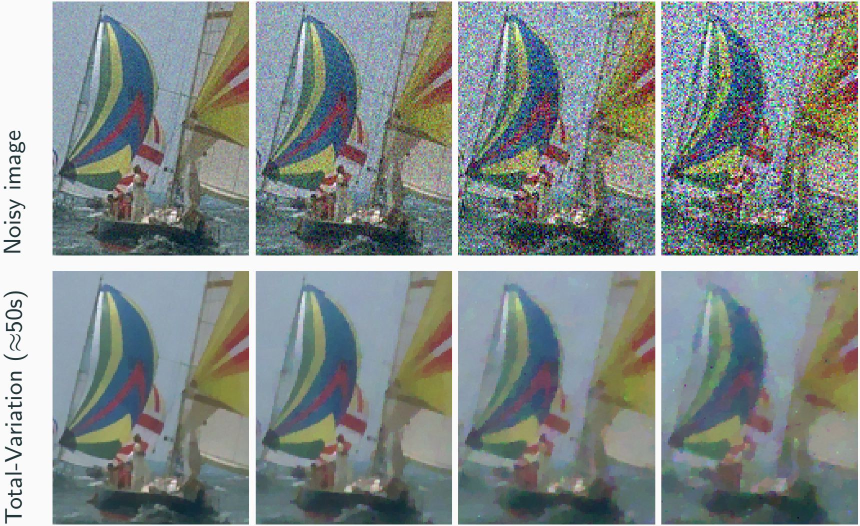
(g) Even larger τ

(h) Too larger τ

Total-Variation – Results

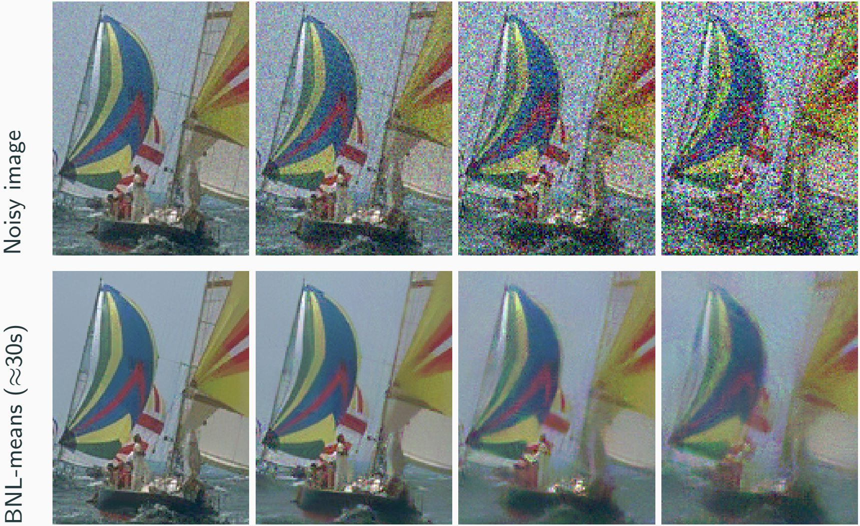
(i) Huge τ

TV regularization for denoising



(a) Noise $\sigma = 10$ (b) $\sigma = 20$ (c) $\sigma = 40$ (d) $\sigma = 60$

TV regularization for denoising



(a) Noise $\sigma = 10$ (b) $\sigma = 20$ (c) $\sigma = 40$ (d) $\sigma = 60$

Questions?

Next class: Introduction to neural networks

Slides from Charles Deledalle and Julie Delon
Sources, images courtesy and acknowledgment

L. Condat

B. Denis de Senneville

A. Horodniceanu

I. Kokkinos

G. Peyré

R. Otazo

V.-T. Ta

Wikipedia