Chapter 2
Linear inverse imaging problems

Variational Methods for Computer Vision
WS 16/17

Michael Moeller
Visual Scene Analysis
Department of Computer Science
University of Siegen

Image Denoising

Denoising

We have seen

Denoising

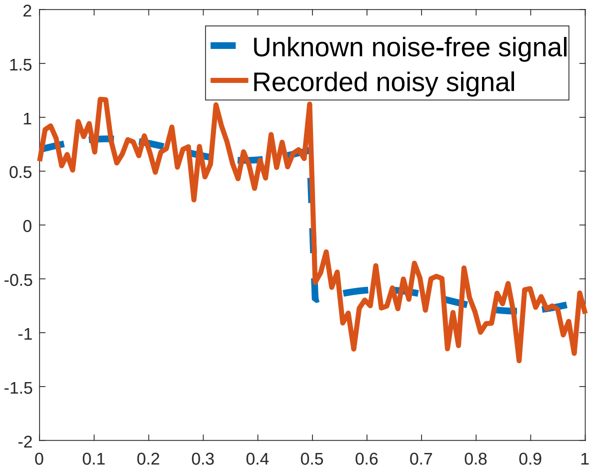
We have seen

Denoising

We have seen

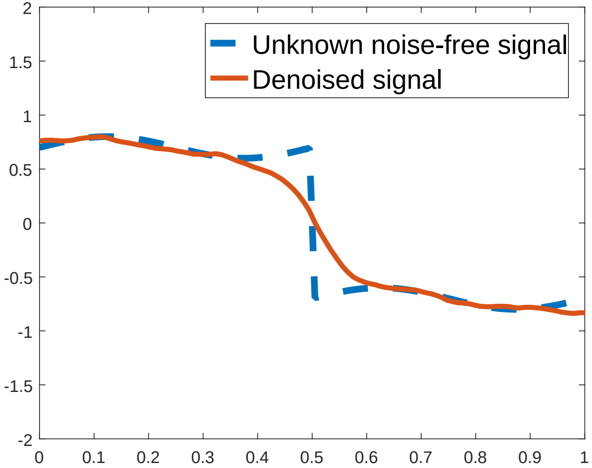
Beyond quadratic ℓ^2-denoising

For signals:



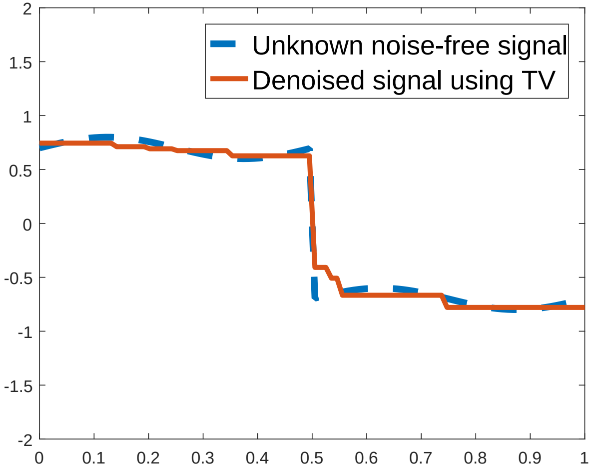
Beyond quadratic ℓ^2-denoising

For signals:



Beyond quadratic ℓ^2-denoising

For signals:



Total variation regularization

For signals we used $\|Du\|_1$, but what shall we use for images?

Total variation regularization (grayscale)

Let $u : \Omega \to \mathbb{R}$, respectively $u \in \mathbb{R}^n$:

isotropic	$\int_\Omega |\nabla u(x)| \, dx$	$\sum_i \sqrt{(D_x u)_i^2 + (D_y u)_i^2}$
anisotropic	$\int_\Omega |\partial_{x_1} u(x)| + |\partial_{x_2} u(x)| \, dx$	$\sum_i |(D_x u)_i| + |(D_y u)_i|$

For the sake of compact writing we will sometimes understand $D : \mathbb{R}^n \to \mathbb{R}^{n \times d}$ by stacking the x- and $y-$derivatives in the second dimension of Du. We write

isotropic $\|Du\|_{2,1}$

where $\|A\|_{1,q} = \sum_i \|A_{i,:}\|_q$.

anisotropic $\|Du\|_{1,1}$

Color total variation regularization

Let $u : \Omega \to \mathbb{R}^c$, respectively $u \in \mathbb{R}^{n \times c}$. Now we have many more options, e.g.

fully anisotropic	$\sum_i \int_\Omega \vert\partial_{x_1} u^i(x)\vert + \vert\partial_{x_2} u^i(x)\vert \; dx$
spatially isotropic, colors anisotropic	$\sum_i \int_\Omega \sqrt{(\partial_{x_1} u^i(x))^2 + (\partial_{x_2} u^i(x))^2} \; dx$
spatially anisotropic, colors isotropic	$\int_\Omega \sqrt{\sum_i (\partial_{x_1} u^i(x))^2} + \sqrt{\sum_i (\partial_{x_2} u^i(x))^2} \; dx$
fully isotropic	$\int_\Omega \sqrt{\sum_i (\partial_{x_1} u^i(x))^2 + (\partial_{x_2} u^i(x))^2} \; dx$

If not stated otherwise, I will use the fully isotropic version and denote it by $\|Du\|_{2,2,1}$ in the discrete setting. [1]

[1] For details: Duran et al. '16, *Collaborative Total Variation: A General Framework for Vectorial TV Models*.

Implementation of total variation regularization

- Since norms are not differentiable, we will smooth them.
- One way of smoothing it is to replace

$$\|d\|_2 = \sqrt{\sum_i d_i^2} \quad \rightarrow \quad \sqrt{\sum_i d_i^2 + \epsilon^2}$$

- Another way of smoothing is the *Huber-loss* $H(z) = \sum_i h_\epsilon(z_i)$ with

$$h_\epsilon(z_i) = \begin{cases} \frac{1}{2} z_i^2 & \text{if } |z_i| \leq \epsilon \\ \epsilon(|z_i| - \frac{1}{2}\epsilon) & \text{else} \end{cases}$$

- For both ways of smoothing one can expect to approximate $\|\cdot\|_2$ more closely as $\epsilon \rightarrow 0$.
- The smaller ϵ, the smaller the descent time step!

ϵ	0.1	0.01	0.001
Avg. τ	0.053	0.009	0.006

Results of total variation regularization

Various denoising results

Original

Results of total variation regularization

Various denoising results

Noisy

Results of total variation regularization

Various denoising results

ℓ^2 squared

Results of total variation regularization

Various denoising results

Smoothed $\epsilon = 0.1$

Results of total variation regularization

Various denoising results

Smoothed $\epsilon = 0.01$

Results of total variation regularization

Various denoising results

Smoothed $\epsilon = 0.001$

Results of total variation regularization

Various denoising results

Smoothed $\epsilon = 0.01$, double-opponent color TV

Results of total variation regularization

Many improvements are based on the idea: Encourage jumps in the color channels to be in the same place

Possible by changing the penalty function of $\nabla u(x)$, or changing the color space of u, e.g. into intensity, hue, saturation.

Example result of the previous slide: *Double-Opponent Vectorial Total Variation* of Aström and Schnörr, ECCV 16:

$$\begin{aligned} TV_{DOVTV}(u) =& TV(u^R) + TV(u^G) + TV(u^B) \\ &+ TV(u^R - u^G) + TV(u^R - u^B) + TV(u^G - u^B) \end{aligned}$$

for $u = (u^R, u^G, u^B)$.

Brief summary

- Images have discontinuities
- ℓ^2 squared regularization, i.e.

$$\hat{u} = \underset{u}{\operatorname{argmin}} \frac{1}{2}\|u - f\|_2^2 + \alpha\|Du\|_F^2$$

blurs the resulting images!

- Replacing $\|\cdot\|_F^2$ by $\|\cdot\|_{2,2,1}$, i.e.,

$$\hat{u} = \underset{u}{\operatorname{argmin}} \frac{1}{2}\|u - f\|_2^2 + \alpha\|Du\|_{2,1,1}$$

gives significantly better results!

Question: What about the data term? Can a penalty different from $\frac{1}{2}\|u - f\|^2$ improve the results even further?

Maximum a-posteriori probability estimates (MAP)

Answer: That depends on the noise we expect f to have!

- Considered measurements f.
- Desired: Noise-free version u.
- Question: What is our best guess?
- Consider noise as a random variable.
- Idea: Maximize $p(u|f)$.

Board: Maximum a-posteriori probability (MAP) estimates.

Maximum a-posteriori probability estimates (MAP)

MAP estimates: Maximize $p(u|f)$

$$\hat{u} = \arg\max_u \; p(u|f) \overset{\text{Bayes}}{=} \arg\max_u \; \frac{p(f|u)p(u)}{p(f)}$$

Now minimize the negative log-likelihood:

$$\begin{aligned}\hat{u} &= \arg\min_u \; -\log\left(\frac{p(f|u)p(u)}{p(f)}\right) \\ &= \arg\min_u \; -\log(p(f|u)) - \log(p(u))\end{aligned}$$

Maximum a-posteriori probability estimates (MAP)

$$\hat{u} = \arg\min_u \; -\log(p(f|u)) - \log(p(u))$$

Question for $p(f|u)$ – noise model. E.g. Gaussian distribution

$$p(f|u) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\|u-f\|^2}{2\sigma^2}\right)$$

Question for $p(u)$ – image prior. E.g. Laplace distribution of the gradient

$$p(u) = \frac{1}{2\beta} \; \exp\left(-\frac{\|Du\|_{2,2,1}}{\beta}\right)$$

Leads to

$$\begin{aligned}\hat{u} &= \arg\min_u \; \frac{1}{2\sigma^2}\|u-f\|^2 + \frac{1}{\beta}\|Du\|_{2,2,1} \\ &= \arg\min_u \; \frac{1}{2}\|u-f\|^2 + \underbrace{\frac{\sigma^2}{\beta}}_{=:\alpha} \|Du\|_{2,2,1}\end{aligned}$$

Maximum a-posteriori probability estimates (MAP)

Other data terms:

- Laplace noise - ℓ^1:

$$\|u - f\|_1$$

- Poisson noise - Kullback Leibler:

$$\sum_i u_i - f_i \log(u_i)$$

- Multiplicative speckle noise:

$$\frac{1}{2\sigma} \sum_i \frac{u_i - f_i}{u_i} + \frac{1}{2} \log(u_i)$$

Example: Removing salt-and-pepper noise

TV-ℓ^1 denoising

Further image priors

We have seen: Adapting the data term based on the MAP estimate

$$\hat{u} = \arg\min_u \; -\log(p(f|u)) - \log(p(u))$$

can greatly improve the results!

Question: Can we obtain further improvements by also adapting $p(u)$?

Much more freedom since the question for ”how likely is an image” is very difficult!

Certain modeling assumption lead to certain regularizations and can improve results.

Next topic: nonlocal regularization!

Self-similarity

One good assumption for many practical applications is that images are self-similar.

Self-similarity

One good assumption for many practical applications is that images are self-similar.

Self-similarity

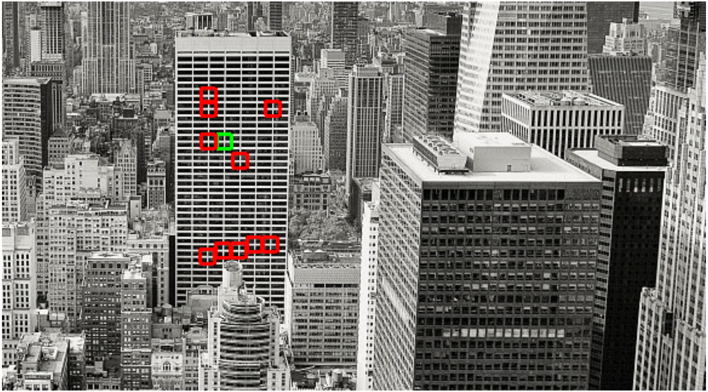
One good assumption for many practical applications is that images are self-similar.

Self-similarity

One good assumption for many practical applications is that images are self-similar.

Self-similarity

One good assumption for many practical applications is that images are self-similar.



Nonlocal filtering

For denoising, one can constrain pixels that lie in similar patches to have similar values!

Let $p_{i,j} \in \mathbb{R}^{w \times w}$ be a patch of the (noisy) input image f of size $w \times w$ centered around pixel (i,j). One determines the similarity matrix W by

$$W_{(i,j),(k,l)} = \exp\left(-\frac{\|p_{i,j} - p_{k,l}\|^2}{\sigma^2}\right).$$

Different options what to do with the similarity matrix. Simple option: **Non-local means**[2].

Normalize weights in W to $\sum_{(l,k)} W_{(i,j),(k,l)} = 1$ for all (i,j). Then determine

$$u_{i,j} = \sum_{k,l} W_{(i,j),(k,l)} f_{k,l}$$

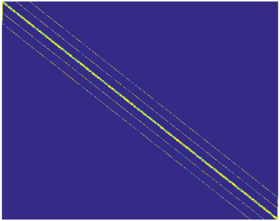
[2] A. Buades, J.-M. Morel. *A non-local algorithm for image denoising*, 2005

Similarity matrices

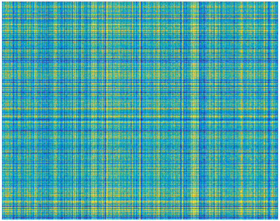
Problem: For $u \in \mathbb{R}^{n \times m}$ one has $W \in \mathbb{R}^{nm \times nm}$!

Two options:

Sparse W, high rank



Dense W, low rank



For sparse matrices:

1. Compute the weights only in an $s \times s$ neighborhood of each (i, j), such that $W \in \mathbb{R}^{nm \times s^2}$.
2. Use greedy methods to find the s (approximate) nearest neighbors

For low rank matrices: use Nystrom extension (discussed later)

Nonlocal means

Illustration for local nearest neighbor search (for a slightly different algorithm)

http://www.cs.tut.fi/~foi/GCF-BM3D/

Example of how NL-means works

http://demo.ipol.im/demo/bcm_non_local_means_denoising/?

Nonlocal means

$$u_{i,j} = \sum_{k,l} W_{(i,j),(k,l)} f_{k,l}$$

is a filter. Is there a variational method behaving similarly?

Nonlocal regularization

Idea: Use

$$R(u) = \frac{1}{4} \sum_{(i,j)} \sum_{(k,l)} W_{(i,j),(k,l)} (u_{i,j} - u_{k,l})^2$$

as a regularization.

For simplicity, let us assume we vectorize our images such that

$$R(u) = \frac{1}{4} \sum_{s} \sum_{t} W_{s,t} (u_s - u_t)^2$$

Let us do one step of gradient descent with step size 1 starting from $u_0 = f$:

$$u_1 = f - \nabla R(f)$$

For a symmetric W we find

$$(\nabla R(u))_l = \sum_t W_{l,t}(u_l - u_t) \overset{\text{if normalized}}{=} u_l - \sum_t W_{l,t} u_t$$

Nonlocal regularization

Our gradient descent step becomes

$$u_1 = f - \nabla R(f) = f - (f - Wf) = Wf$$

One step of gradient descent is the same as nonlocal means!

Particularly interesting:

$$\nabla R(u) = \underbrace{u}_{\text{current point}} - \underbrace{Wu}_{\text{Average of (nonlocal) neighborhood}}$$

Particularly interesting:

$$\nabla R(u) = \underbrace{u}_{\text{current point}} - \underbrace{Wu}_{\text{Average of (nonlocal) neighborhood}} = -\underbrace{(W - I)}_{\text{Graph Laplacian}} u$$

Our very first regularization is a special case of the above framework for $W_{i,i-1} = W_{i,i+1} = 0.5$.

$$J(u) = \frac{1}{4}\|Du\|_2^2 = \frac{1}{4}\sum_i (u_i - u_{i-1})^2 \overset{\text{corresponds to}}{\rightarrow} c \cdot \|\nabla u\|_2^2$$

The derivative (away from the boundary) is

$$(\nabla J(u))_i = u_i - \frac{u_{i-1} + u_{i+1}}{2} \overset{\text{corresponds to}}{\rightarrow} -\Delta u$$

Graph Laplacian

For $u : \mathbb{R}^d \to \mathbb{R}$ it holds that[3]

$$\Delta u(x) = \lim_{\epsilon \to 0} \frac{2d}{c(\epsilon)} \left(\text{Average}_{N_\epsilon(x)}(u) - u(x) \right)$$

We can interpret a similarity matrix W of an image as a graph! Neighborhoods are defined by those points that have large similarity values (strong edges).

To normalize the matrix W let us define $D = \text{diag}(d_i)$ for

$$d_i = \sum_j W_{i,j}$$

$\to$ The operator $(D^{-1}W - I)$ provides the notion of a Laplacian operator on a graph!

---

[3] See talk by Peyman Milanfar `https://www.pathlms.com/siam/courses/2426/sections/3234` for details.

Graph Laplacian

Unfortunate: $L = D^{-1}W - I$ is not symmetric (and called *random walk Laplacian*).

Some alternative proposals for graph Laplacians

Unnormalized Laplacian	$L = W - D$
Normalized Laplacian	$L = D^{-1/2}WD^{-1/2} - I$
Sinkhorn Laplacian	$L = C^{-1/2}WC^{-1/2} - I$

All the above versions are symmetric and negative semi-definite.

With any version we can use

$$R(u) = -\frac{1}{2}\langle u, Lu\rangle$$

as a regularization.

Implementation details - sparse similarity matrices

Typical procedure for computing W on an image $f \in \mathbb{R}^{n \times m}$.

1. Fix a search window size s around each pixel. The sparse matrix $W \in \mathbb{R}^{nm \times nm}$ has at most $(2s+1)^2 \cdot nm$ many entries.

2. Fix a patch size p, i.e. one computes

$$W_{(i,j),(k,l)} = \exp\left(- \sum_{r=-p}^{p} \sum_{t=-p}^{p} (f_{i+r,j+t} - f_{k+r,l+t})^2 / h \right)$$

for a scaling factor h

Trick to make the computation of the $(2p+1)^2 \cdot (2s+1)^2 \cdot nm$ many terms fast: *Integral images.*

Integral images

For each of the $(2s+1)^2$ many shifts (w, v)

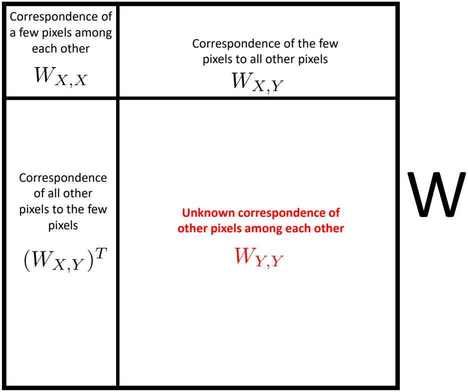
- Shift f by (w, v) and denote the shifted version by g
- Compute $e = (f - g).^2 \leftarrow$ pointwise
- Compute an integral image I in which the (i, j)-th entry is the sum of all upper left pixels of e
- Patch-wise differences can be computed by evaluating only 4 points of I

See illustration for pictures, do exercise for formulas!

Implementation details - low rank similarity matrices

The previous technique yields a high rank sparse matrix W.

Alternative: Dense low rank matrix. Common way to obtain this: Nyström extension.



One sets

$$W_{Y,Y} = (W_{X,Y})^T (W_{X,X})^{-1} W_{X,Y}$$

Nyström method

For $W \in \mathbb{R}^{N \times N}$ symmetric positive semi-definite we can write $W = B^T B$. Let us write $B = \begin{pmatrix} X & Y \end{pmatrix}$ for $X \in \mathbb{R}^{N \times l}$, $Y \in \mathbb{R}^{N \times N-l}$. We find

$$B^T B = \begin{pmatrix} X^T \\ Y^T \end{pmatrix} \begin{pmatrix} X & Y \end{pmatrix} = \begin{pmatrix} X^T X & X^T Y \\ Y^T X & Y^T Y \end{pmatrix}$$

The approxmation done by the Nystrom method is to set

$$Y^T Y \approx Y^T X (X^T X)^{-1} X^T Y$$

For a more detailed derivation using eigenvector theory see for instance Fowlkes et al., *Spectral Grouping Using the Nyström Method*, PAMI 2004.

Nyström method

An approximate eigendecomposition of W is given as

$$W = \bar{U}\,\Sigma\,\bar{U}^T, \qquad \text{for} \quad \bar{U} = \begin{pmatrix} U \\ (W_{X,Y})^T U \Sigma^{-1} \end{pmatrix} \in \mathbb{R}^{N\times r}$$

for $\Sigma \in \mathbb{R}^{r\times r}$ denoting the diagonal matrix with the eigenvalues of $W_{X,X} \in \mathbb{R}^{r\times r}$ on the diagonal and U being the corresponding eigenvectors.

Advantages of the Nyström method:

- We can store a dense $W \in \mathbb{R}^{N\times N}$ by storing $N \cdot n + n^2 << N^2$ values.
- Multiplication with W is cheap by carrying out 2 or 3 separate small matrix multiplications.
- Exercise: For W being of low rank and having rows that sum to one, there is an efficient formula for solving

$$\hat{u} = \operatorname*{argmin}_u \frac{1}{2}\|u - f\|^2 + \frac{\alpha}{2}\langle u, (I - W)u\rangle$$

Intermediate summary

Nonlocal methods construct a similarity matrix $W \in \mathbb{R}^{nm \times nm}$ on the input image $f \in \mathbb{R}^{n \times m}$

The nonlocal means filter computes a denoised version of f as $D^{-1}Wf$ where D is the diagonal matrix containing the row sums of W on the diagonal.

$L = D^{-1}W - I$, $L = W - D$, and $L = D^{-1/2}WD^{-1/2} - I$ are possible variants for defining graph Laplacians.

Nonlocal regularization penalizes

$$R(u) = -\frac{1}{2}\langle u, Lu\rangle$$

in full analogy to our first simple local regularization

$$R(u) = -\frac{1}{2}\langle u, \Delta u\rangle = \frac{1}{2}\|\nabla u\|^2$$

The above methods become computationally feasible by either making W sparse or low rank (using Nyström's method).

Rigorous treatment of nonlocal operators

We call

$$TV_I^W(u) = \int_\Omega \sqrt{\int_\Omega (u(y) - u(x))^2 w(x, y)\; dy\; dx}$$

$$TV_A^W(u) = \int_\Omega \int_\Omega |u(y) - u(x)| \sqrt{w(x, y)}\; dy\; dx$$

the isotropic and anisotropic nonlocal total variation, respectively. [4]

Discrete variants

$$TV_I^W(u) = \sum_{i,j} \sqrt{\sum_{k,l} |u_{i,j} - u_{k,l}|^2 W_{(i,j),(k,l)}}$$

$$TV_A^W(u) = \sum_{i,j} \sum_{k,l} |u_{i,j} - u_{k,l}| \sqrt{W_{(i,j),(k,l)}}$$

[4] General framework from Gilboa, Osher, *Nonlocal operators with applications to image processing*, 2008.

Problems of nonlocal regularizations

Relies on the assumption that images are self-similar.

Requires tuning of the internal paramters (e.g. search window, patch size, scaling factor).

One would ideally like to use

$$W_{(i,j),(k,l)}(u) = \exp\left(-\sum_{r=-s}^{s}\sum_{t=-s}^{s}(u_{i+r,j+t} - u_{k+r,l+t})^2/h\right)$$

in the optimization for u. Such problems would be extremely difficult to optimize. Using $u = f$ instead assumes that the data is ”good enough” to get reliable estimates.

Summary Denoising

We have seen several ways to formulate image denoising as

$$\hat{u} = \underset{u}{\operatorname{argmin}}\, H_f(u) + \alpha R(u)$$

The data fidelity term H_f relates the solution u to the measured data f. We can adapt it to the expected type of noise using MAP estimates.

We have learned about several types of regularizations

1. Local quadratic ℓ^2 regularzation, $\|\nabla u\|^2$.
2. TV regularization, e.g. $\|\nabla u\|_{2,1}$.
3. Nonlocal quadratic regularization.
4. Nonlocal TV regularization.

Great thing about variational methods: We will move on to another application, but you can keep using the same types of regularizations!

Image Deblurring

Removing blurs from images

An *image blur* is a (subjective) describtion of an image not being sharp / looking unnatural.

Gaussian blur ($\rightarrow$ out of focus)

Removing blurs from images

An *image blur* is a (subjective) describtion of an image not being sharp / looking unnatural.

Motion blur

Removing blurs from images

An *image blur* is a (subjective) describtion of an image not being sharp / looking unnatural.

Different motion blur

Image sharpening using linear filters

We already learned about one simple technique to sharpen an image.

$$\hat{u} = (I - \alpha\Delta) f$$

Image sharpening using linear filters

We already learned about one simple technique to sharpen an image.

$$\hat{u} = (I - \alpha\Delta)f$$

Image sharpening using linear filters

We already learned about one simple technique to sharpen an image.

$$\hat{u} = (I - \alpha\Delta)f$$

Problems:

- Works for small undirected blurs only.
- Heavily amplifies noise.
- Neglects information we possibly have about the blur.

Modelling blurs

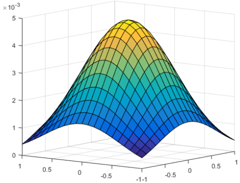
How can we model a blur?

Most common: **Convolution**

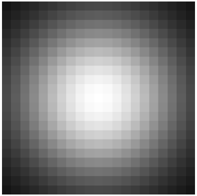
Cont.: $$(k * u)(x_1, x_2) = \int_\Omega k(y_1, y_2) u(x_1 - y_1, x_2 - y_2)\, dy_1\, dy_2$$

Disc.: $$(k * u)(i, j) = \sum_{s,t} k(s, t)\, u(i - s, j - t),$$

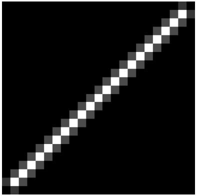
where k is called convolution kernel in the discrete and continuous setting and sometimes mask in the discrete.
Common assumption in practice: k has finite support



Truncated Gaussian



Truncated Gaussian



Motion blur kernel

Convolutions

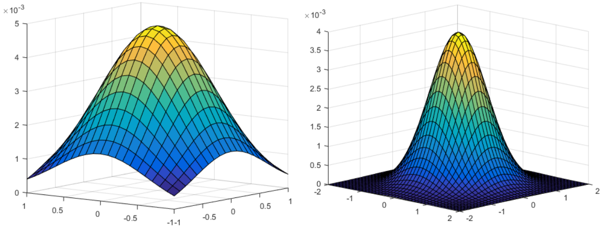
Discrete convolutions are a local weighted average in a neighborhood or window around each pixel:

$$\text{Disc.:} \quad (k * u)(i,j) = \sum_{s=-h}^{h} \sum_{t=-w}^{w} k(s,t)\, u(i-s, j-t)$$

Here $(2h+1) \times (2w+1)$ window.

Illustration on
https://en.wikipedia.org/wiki/Convolution

Save computations by truncating the support



Convolutions

We have already worked with discrete convolutions!

The convolution with

$1/h \cdot$

-1	1	0

yields an approximation to the *x*-derivative. The transpose kernel yields a *y*-derivative.

The convolution with

$1/h^2 \cdot$

0	1	0
1	-4	1
0	1	0

yields an approximation to the Laplace operator.

Deblurring becomes Deconvolution!

Back to the problem of how to remove the blur from images under a particular assumption on the data formation process:

$$f = k * u_{\text{true}} + \text{noise}$$

Again, we can compute the MAP estimate and find - for example under the assumption that the noise is Gaussian - we use

$$\frac{1}{2}\|k * u - f\|^2$$

as a data fidelity term.

Remember from linear algebra: The application of any linear operator in finite dimension (such as $k * u$) can be represented as a matrix vector multiplication.

Writing the convolution as a matrix multiplication

We already learned one way of obtaining a matrix vector representation: For separable kernels, $k(x, y) = k_1(x)k_2(y)$, we can find matrices K_x and K_y such that

$$k * u = K_y u K_x$$

and find

$$\text{vec}(k * u) = \text{kron}(K_x^T, K_y)\text{vec}(u).$$

$\rightarrow$ ***Small code demonstration.***

Easy alternative (with zero boundary conditions and a larger convolution result): MATLAB's *convmtx2*.

Minimizing the data term only?

After we have written the convolution as a vector matrix multiplication, we just aim at solving a linear system approximately:

$$Ku \approx f \qquad \text{or } \|Ku - f\|^2 \text{ small.}$$

What happens if we do gradient descent on the data term?

Input data (iteration 0)

Minimizing the data term only?

After we have written the convolution as a vector matrix multiplication, we just aim at solving a linear system approximately:

$$Ku \approx f \qquad \text{or } \|Ku - f\|^2 \text{ small.}$$

What happens if we do gradient descent on the data term?

Iteration 40

Minimizing the data term only?

After we have written the convolution as a vector matrix multiplication, we just aim at solving a linear system approximately:

$$Ku \approx f \qquad \text{or } \|Ku - f\|^2 \text{ small.}$$

What happens if we do gradient descent on the data term?

Iteration 2000

Ill-posed inverse problem

One can show: That all problems of recovering u the form measurements

$$Au(x) = \int_\Omega k(x,y)u(y)\ dy$$

with kernel $k \in L^2(\Omega \times \Omega)$. Leads to an **ill-posed** problem because **the solution does not depend on the data continuously**.

All kinds of regularization we developed for image denoising can immediately be put in use!

$$\hat{u} = \min_u \frac{1}{2}\|Ku - f\|^2 + \alpha R(u),$$

e.g. for $R = \frac{1}{2}\|\nabla u\|^2$, $R = \|\nabla u\|_{2,1}$, or nonlocal regularizations such as $R = \langle u, Lu \rangle$ for the graph Laplacian L.

Implementation of Deblurring

TRY IT:

- We discussed how to generate K
- You know how to generate $L = I - D^{-1/2} W D^{-1/2}$
- You have code to generate W.
- You have code for computing (approximations to) $R(u) = \|\nabla u\|_{2,1}$, and $\nabla R(u)$.

Your task on the next exercise sheet: Implement it!

Any questions about deblurring and it's implementation?

An Example

Removing a known motion blur

Blurry and noisy input image

An Example

Removing a known motion blur

Straight forward TV regularized reconstruction

Some remarks

Interesting recent variant for the data term: Also compare the local or nonlocal gradients in the data term, i.e. use

$$\frac{1}{2}\langle Ku - f, (I - \beta L)Ku - f) \rangle$$

for L being the (possibly nonlocal) Laplace operator.

There are methods trying to also estimate the noise standard variation to adapt the regularization parameter α in

$$\min_u H_f(u) + \alpha R(u)$$

Nonlocal regularization methods sometimes iteratively adapt the similarity matrix W, see e.g. Kheradmand and Milanfar, *A general framework for regularized, similarity-based image restoration*, 2014.

We considered the case where the blur operator K is known. Jointly estimating K and u is called **blind deconvolution** and will be discussed later.

An Example

Removing a Gaussian blur - Kheradmand and Milanfar Code

Blurry and noisy input image

An Example

Removing a Gaussian blur - Kheradmand and Milanfar Code

Nonlocal reconstruction

Punchline

Whenever we know the (linear) data formation process, we can use our known regularizations R to solve a variational problem of the form

$$\frac{1}{2}\|Au - f\|^2 + \alpha R(u)$$

for application-dependent linear operators A!

Based on the type of noise we expect, we can adapt the norm for the data fidelity term.

Image Zooming

https://www.youtube.com/watch?v=LhF_56SxrGk

Image zooming

Consider our favorite image

Original books image

Image zooming

But we observed only a low resolution version

How do we go back to a high resolution version?

Image zooming

Repeating pixel values

Nearest Neighbors interpolation

Image zooming

High order interpolation

Bicubic upsampling

Image zooming

Respecting the assumed data formation process

Variational method

Image zooming

As soon as we understand the forward model, obtaining the results on the previous slide is easy!

Forward model = Given the high resolution version, how would you create a low resolution version?

Steps:

1. Blur the high resolution image (to avoid aliasing)
2. Average the values of the high resolution pixels to obtain the low resolutino pixel value.

Define $A = D\ B$ for a blur operator B and a downsampling operator D and solve

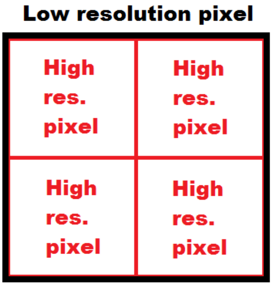
$$\frac{1}{2}\|Au - f\|^2 + \alpha R(u)$$

→ **Variational methods are awesome!!**

The downsampling operator

How would you model a downsampling operator?

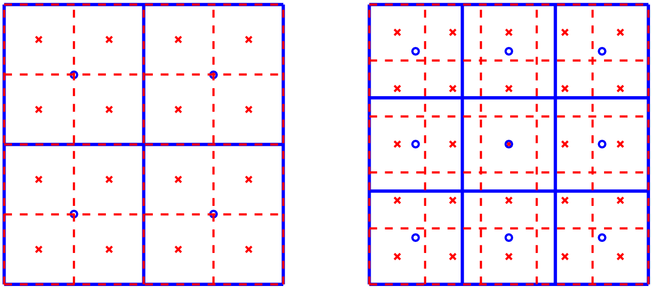
Example: Subsampling by a factor of 2:



Each low resolution pixel should be the mean of the corresponding high resolution pixels.

The downsampling operator

Making things more complicated:



Interpolation 4×4 to 2×2 Interpolation 5×5 to 3×3

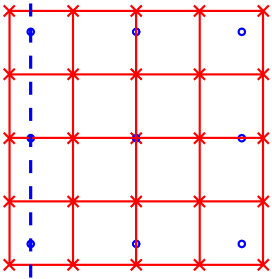
Possible approach to generate a forward model: The values at the blue (low resolution) pixels originate from the red (high resolution) pixels via a bilinear interpolation.

Reminder: Bilinear interpolation

Explain bilinear interpolation on the board.

We always represent images on a uniform grid.

$\rightarrow$ The x-interpolation is the same for each row of of the image!



$\rightarrow$ We can write the x-interpolation as uS_x for u being the image in matrix form, and S_x being a suitable interpolation matrix.

Final remarks

The problem

$$\hat{u} \in \underset{u}{\operatorname{argmin}} \frac{1}{2}\|DBu - f\|^2 + \alpha R(u)$$

can be solved as usual, i.e.

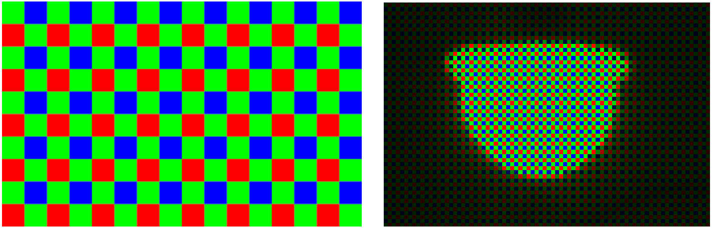
1. Solve a linear system (e.g. with pcg in MATLAB) if R is quadratic.
2. Use our line search gradient descent algorithm if R contains a (smoothed) ℓ^1 type norm.

Keep in mind: in many cases (e.g. upsampling by an integer factor) generating D in matrix form is highly inefficient!! It is just a tool for us to have a linear operator that is easy to deal with and to show that many problems are structurally the same!!

Image Demosaicking

Image demosaicking

Color filter arrays (CFAs): The way a camera records colors



Most common: Bayer pattern

Forward model:

$$Au(x) = \begin{cases} u(x) & \text{if color known,} \\ 0 & \text{if color not known.} \end{cases}$$

Inpainting problem:

$$u(\alpha) = \arg\min_u \frac{1}{2}\|A(u-f)\|_2^2 + \alpha R(u)$$

Image demosaicking

Consideration on the board: Separate channel TV regularization leaves the problem underdetermined!

Nonlocal methods are difficult to apply.

Example for a variational demosaicking approach: Pascal Getreuer, *Color Demosaicing with Contour Stencils*, 2011.

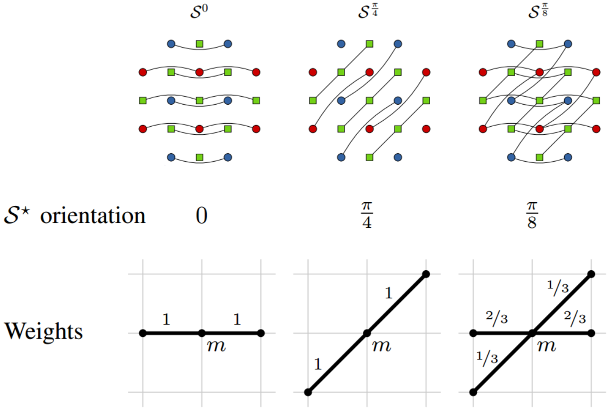


Image demosaicking

Strategy:

- Determine the best fitting stencil at each pixel.
- Assign weights for connecting the differences of pixel values at each location.
- Smooth weights and make them symmetric $\to w_{i,j}$.
- Compute directed total variation minimizing solution, i.e.

$$\min_u R(u) = \sum_i \sqrt{\sum_j w_{i,j}(u_i - u_j)^2} \quad \text{s.t. } Au = Af$$

or a denoising-based alternative

$$u(\alpha) = \arg\min_u \frac{1}{2}\|A(u-f)\|_2^2 + \alpha R(u)$$

An alternative regularization?

What are contour stencils trying to do?

From a modeling perspective they argue we should EITHER smooth horizontally OR vertically OR diagonally at each pixel. Which direction to use is determined beforehand.

This raises a quite general question: How do I model logical combinations of regularization terms?

Easy: R_1 AND R_2 should be small:

$$R(u) = R_1(u) + R_2(u)$$

Difficult: R_1 OR R_2 should be small. Possible modeling

$$R(u) = \min(R_1(u), R_2(u))$$

is highly nonconvex!

Minimum of two regularizations

Possible ways to handle

$$R(u) = \min(R_1(u), R_2(u)) \tag{1}$$

1. Determine weights w_1, w_2 from the input data and use

   $$w_1 R_1(u) + w_2 R_2(u)$$

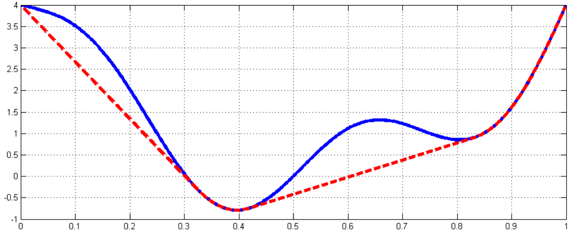
   Drawback: Two-step procedure. Fails if estimation of weights fails. Possibly not robust to noise. Requires data to allow the estimation of weights.

2. Iterative reweighting procedures: Iterate between determining a solution u^k and determining new weights $w_1^k R_1(u) + w_2^k R_2(u)$. Drawback similar to the above. Final result depends on initialization.

3. **Convex relaxation:** Try to find a convex function that approximates (1).

Convex relaxation

Common strategy for convex approximations: Find the largest lower semi-continuous (lsc) convex function conv(E) that underestimates the energy E.Nice property:



A global minimizer of E is a global minimizer of conv(E)!

Convex relaxation

But how can we compute conv(E)?

Definition: Convex Conjugate

Let $E : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be an arbitrary energy function. We call

$$E^*(p) = \sup_u \langle u, p \rangle - E(u)$$

the *convex conjugate* of E.

Biconjugate

The *biconjugate* E^{**} is the largest lsc convex underapproximation of E.

How can we understand this in more detail?

Convex conjugates

Properties of the conjugates

It holds that ...

1. ... the convex conjugate E^* of any function E is convex.
2. ... $E^{**}(u) \leq E(u)$ for all u.
3. ... $E^{**} = E$ if E is convex and lsc.
4. ... conjugation reverses inequalities, i.e.

$$E_1 \leq E_2 \quad \Rightarrow \quad (E_1)^* \geq (E_2)^*$$

-

Proofs: 1) + 2) in the exercise, 3) only partially, i.e. $E^{**}(u) = E(u)$ for E being differentiable at u, on the board.

Now we can easily argue why E^{**} is the largest convex, lsc underapproximation of E.

Fenchel-Rockafellar duality

One of the most central theorems in the field of convex conjugation is

Fenchel-Rockafellar duality theorem

Let $E_1 : \mathbb{R}^n \to \mathbb{R}$ and $E_2 : \mathbb{R}^m \to \mathbb{R}$ be convex, and $A : \mathbb{R}^{m \times n}$ a matrix. Then

$$\inf_u E_1(u) + E_2(Au) = -\inf_p E_1^*(-A^T p) + E_2^*(p).$$

We will exploit (a generalized version of) this theorem heavily in next semester's lecture on convex optimization. For now, consider it as a tool to determine convex conjugates of sums.

Convex conjugates

This is a nice theory (of which we only discussed a tiny, tiny fraction. How do put this into application? What about our $\min(R_1, R_2)$ problem?

You will use everything we discussed to show in the exercises:

Biconjugate of minimum of one-homogeneous functions

Let $R_1 : \mathbb{R}^n \to \mathbb{R}$ and $R_2 : \mathbb{R}^n \to \mathbb{R}$ be convex and absolutely one-homogeneous, i.e.

$$R_1(\alpha u) = |\alpha| R_1(u) \qquad \forall \alpha,\ u.$$

Then the biconjugate of

$$E(u) = \min(R_1(u), R_2(u))$$

is the *infimal convolution*

$$E^{**}(u) = \inf_w R_1(u - w) + R_2(w)$$

between R_1 and R_2.

A toy example for image demosaicking

Let us put everything into use. Assume we believe that a good regularization for demosaicking penalizes either mostly the x-derivative, or mostly the y-derivative.

We define

$$R_1(u) = \int_\Omega \sqrt{(0.25 \cdot \partial_{x_1} u(x))^2 + (\partial_{x_2} u(x))^2} \, dx,$$
$$R_2(u) = \int_\Omega \sqrt{(\partial_{x_1} u(x))^2 + (0.25 \cdot \partial_{x_2} u(x))^2} \, dx.$$

Then we use the the *infimal-convolution* of R_1 and R_2 as a regularization, i.e.

$$R(u) = \min_w R_1(u - w) + R_2(w)$$

A toy example for image demosaicking

Complete demosaicking problem

$$\min_{u,w} \frac{1}{2}\|Au - f\|_2^2 + R_1(u - w) + R_2(w).$$

Algorithmically, we solve for u and w simultaneously! After smoothing R_1 and R_2, we can still apply the same gradient descent method that we always used!

Discuss such an implementation on the board. This will be your next task!

What you should get

What you should get

What you should get

What you should get

Summary

For modeling a convex regularization function that behaves somewhat similar to the minimum of two convex one-homogeneous functions one can use the infimal-convolution

$$(H \Box R)(u) = \inf_{w} H(u - w) + R(w).$$

We briefly introduced the idea of the convex conjugate E^* of E as

$$E^*(p) = \sup_{p} \langle p, u \rangle - E(u),$$

and learned about some computational rules.

We found the biconjugate E^{**} to be the largest convex lsc underapproximation of E.

A whole branch of research investigates the question of how to find good convex approximations to nonconvex cost functions.

Small Excursion: Image formation process!

We have talked about noise levels and considered data fidelity terms like

$$\frac{1}{2}\|Au - f\|_2^2 \quad \text{or} \quad \|Au - f\|_1,$$

but how does it look in practice?

Following information based on code of publication: Andriani et al., *Beyond the Kodak Image Set: A New Reference Set of Color Image Sequences*, 2013.
`ftp://imageset@ftp.arri.de`, password: imageset.

-> Lets look inside!

Image formation process

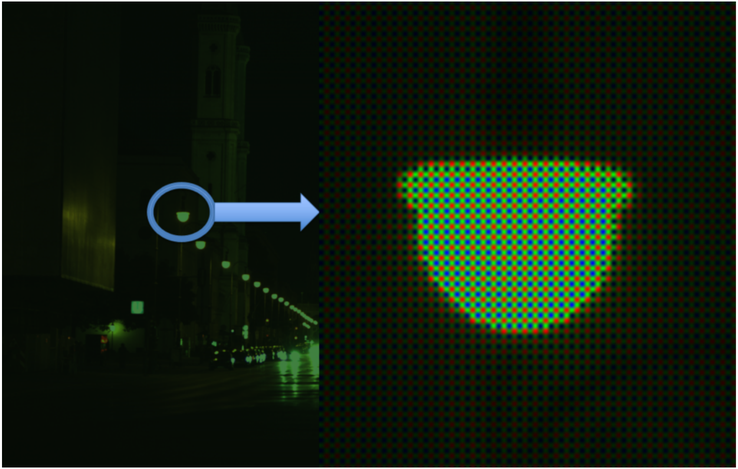


Image formation process

Image formation process

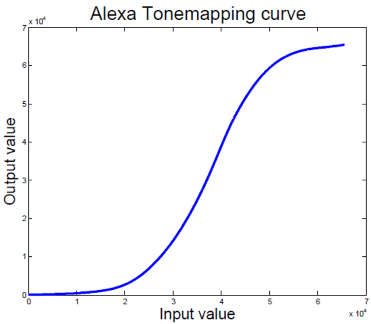
Human visual system: Linear changes in light intensity do not correspond to a linear change in brightness!

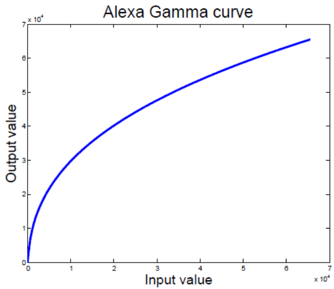
Image formation process

Human visual system: Linear changes in light intensity do not correspond to a linear change in brightness!

Image formation process

Human visual system: contrasts are stronger for medium intensities





Human visual system: Much more sensitive to dark values

Image formation process

Image formation process

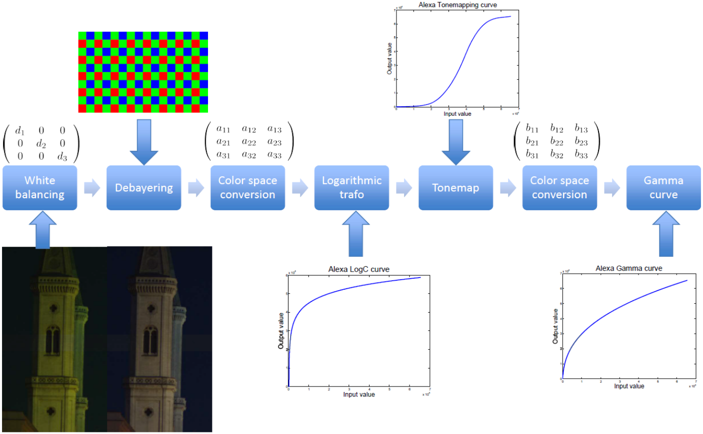
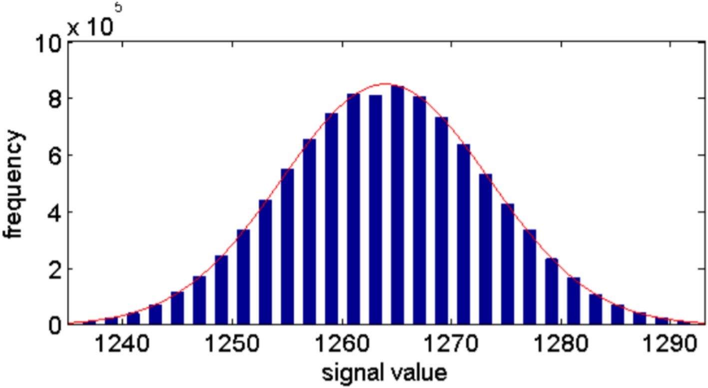
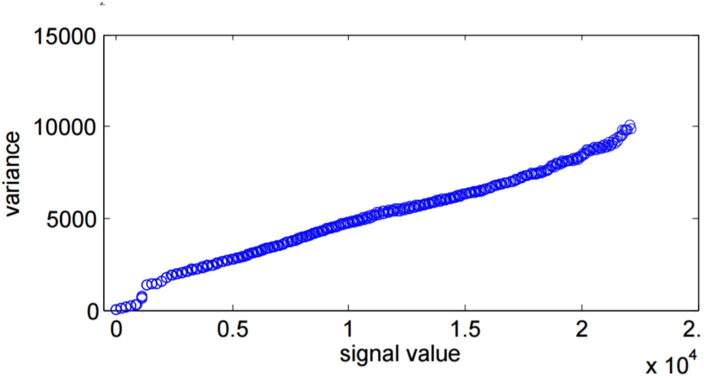


Image formation process



From:. Seybold et al. *Towards an Evaluation of Denoising Algorithms with Respect to Realistic Camera Noise*, 2013.

Image formation process

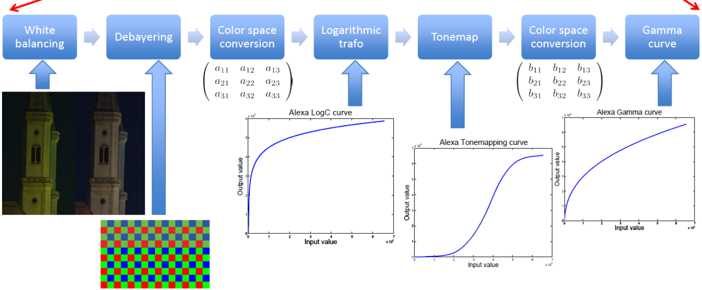


From: Seybold et al. *Towards an Evaluation of Denoising Algorithms with Respect to Realistic Camera Noise*, 2013.

Image formation process

$$\hat{u} = \arg\min_u \left\{ \frac{1}{2}\left\| \frac{u-f}{\sigma} \right\|^2 + \alpha \|\nabla u\|_1 \right\}$$

With a signal dependent and therefore spatially varying sigma that needs to be estimated



The theory of MAP estimates does provide a reasonable data fidelity term, but usual priors are developed on "monitor images". Open question: Best way to deal with this? [5]

[5] Also see Khashabi et al., *Joint Demosaicing and Denoising via Learned Non-parametric Random Fields*, 2013.

Small summary

As as soon as we have a (linear) forward model $f = A\hat{u}$ we immediately know how to formulate a variation method that tries to recover $\hat{u}$, via

$$\underset{u}{\operatorname{argmin}}\, H(Au - f) + \alpha R(u).$$

The data fidelity term can be motivated via MAP estimates.

Excursion: In practice the right MAP estimate is often difficult to obtain.

Some (heuristic) robust cost function (such as ℓ^1) is a decent choice for many applications.

Image Inpainting

General inpainting and image restoration

We have seen: By choosing the linear operator A in

$$\operatorname*{argmin}_u H(Au - f) + \alpha R(u)$$

to "switch off" the data term locally, we can model demosaicking.

Natural question: Can we switch off different/larger parts and fill them with our regularization?

Simply use

$$(Au)_i = \begin{cases} u_i & \text{if } i \in I \\ 0 & \text{else.} \end{cases}$$

for an index set I of our choice!

Random inpainting

Let us choose I to be a certain percentage of random pixels.

First observation: Nonlocal methods or variants like contour stencils be very difficult to use. Unclear how to compute weights!

Regularizations like the total variation are straight forward. Let's try it!

Random inpainting

Original image

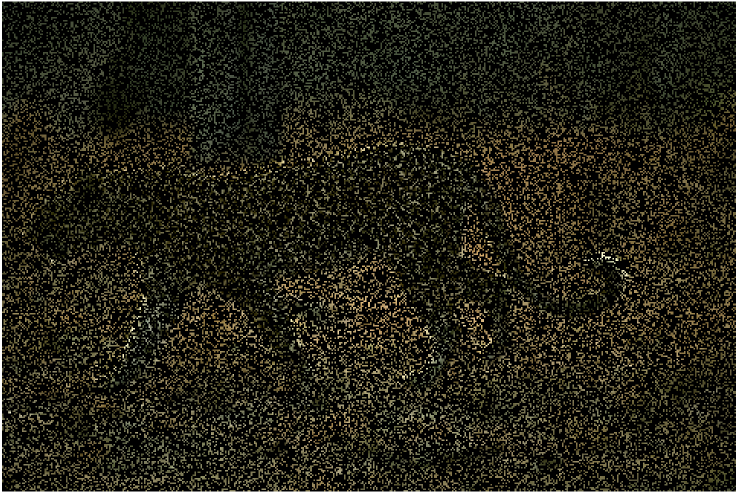
Random inpainting

50% of the pixels missing

Random inpainting

50% of the pixels inpainted

Random inpainting

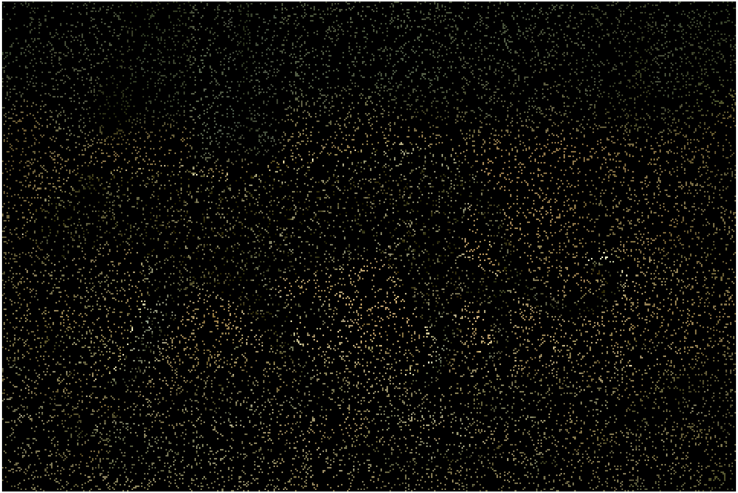


70% of the pixels missing

Random inpainting

70% of the pixels inpainted

Random inpainting



90% of the pixels missing

Random inpainting

90% of the pixels inpainted

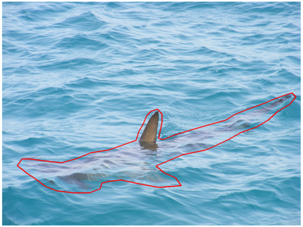
Random inpainting

Research in this direction (but not with fully random pixels): Image compression. See e.g. Mainberger, Weickert, *Edge-based image compression with homogeneous diffusion.* as an example.

Very interesting for image editing: Region inpaining.

We will discuss two examples: Inpainting with and without additional information.

Poisson editing



How do we get the shark into the pool?

Assume we have the contours (e.g. user given).

Straight forward: Copy and paste.

Poisson editing

Looks strange – colors and lighting do not match.

Poisson editing

Better – Blending using Poisson editing

Poisson editing

Poisson editing (Perez, Gangnet, Blake, 2004): Determine fusion by matching gradients.

Local, quadratic inpainting without additional information:

$$\min_u \|\nabla u\|_2^2 \qquad \text{such that } \; u(x) = f(x) \; \forall x \notin M$$

where M is the inpainting domain. Local, quadratic inpainting without additional information:

$$\min_u \|\nabla u {\color{red}- \nabla g}\|_2^2 \qquad \text{such that } \; u(x) = f(x) \; \forall x \notin M$$

where M is the inpainting domain.

Poisson editing

Further remarks

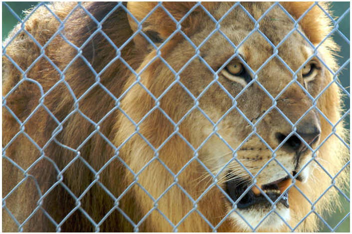
- Instead of using ∇g directly, design the gradient map, e.g. $\alpha \nabla g + \beta \nabla f$, or $\max(\nabla f(x), \nabla g(x))$.
- Smooth the mask and use linear blending at the boundaries:

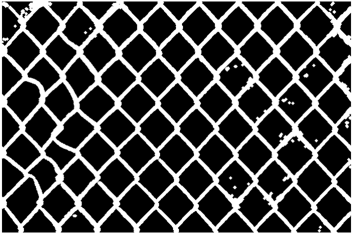
```
smoothMask = imfilter(mask, ones(c)/c^2);
blended = smoothMask.*img1+(1-smoothMask).*img2;
```

- Improvements e.g. "Linear Osmosis Models for Visual Computing" Weickert et al., 2013.

Inpainting based on self-similarity

Can we free the lion without any additional information?





We turn back to our standard inverse problem formulation:

$$\min_u \frac{1}{2}\|Au - f\|_2^2 + \alpha R(u)$$

$$\text{with } Au(x) = \begin{cases} u(x) & \text{if color known,} \\ 0 & \text{if color not known.} \end{cases}$$

Inpainting based on self-similarity

Inpainting based on self-similarity

Higher order inpainting

It works but cannot reconstruct any texture. Thus it looks unrealistic.

Improvements can be made by considering higher order methods such as the curvature based inpainting by Masnoi, Morel, *Level-lines based disocclusion*

Also see Sapiro, Bertalimio, Casseles, Ballester, *Image Inpaining* for an interesting PDE based approach, and follow up papers for further discussions.

We will yet discuss a different approach that exploits self-similarity of images in a different way than the nonlocal regularizations we have seen before.

Regularization via dictionary learning

Self-similarity revisited

We already learned about non-local means and non-local regularization, which exploid the fact that for each patch in the image there exist several similar patches.

With 90% of the pixels missing it is difficult to identify similar patches. For arbitrary linear operators A in

$$\|Au - f\|_2^2$$

it is unclear how to apply non-local methods.

Idea: Use training data to generate a **dictionary** of patches to compare to!

Sparse representation

Maintaining a dictionary that contains similar patches for all possible images a person could ever record / try to reconstruct would be too large.

Modeling assumption: One can learn a dictionary $D \in \mathbb{R}^{n \times N}$, $N >> n$, such that an arbitrary patch $p \in \mathbb{R}^n$ in a natural image can be represented as a **sparse linear combination** of atoms of D:

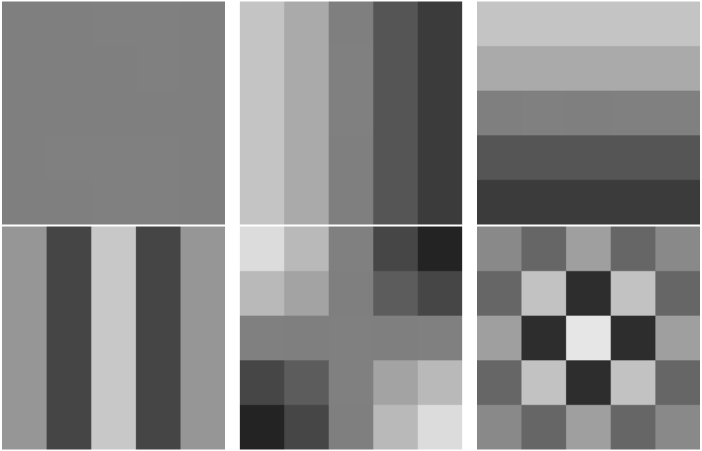
$$p = D\alpha, \qquad |\alpha|_0 \text{ small},$$

where the ℓ^0 "norm" $|\alpha|_0$ (which is not a norm) is the number of nonzero entries in α.

Very common and powerful assumption even for designed dictionaries such as wavelets or DCT transform and for instance used in image compression.

DCT compressibility

Example DCT basis elements on 5×5 patches



Patchwise representation of the image already yields decent sparse representation!

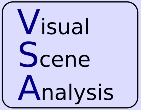
By *learning* an *overcomplete* basis, we can expect to get significantly sparser representations.

DCT compressibility

Original image - 98% nonzero DCT coefficients

DCT compressibility

18% nonzero DCT coefficients



DCT compressibility

6.2% nonzero DCT coefficients

DCT compressibility

4.7% nonzero DCT coefficients

Sparse coding

Assume we are given a number of natural images and extracted a large number of M patches, which - when stacked as a vector - have size n. We form a matrix $Y \in \mathbb{R}^{n \times M}$ which is our **training data**.

Formalizing our problem: Find a dictionary $D \in \mathbb{R}^{n \times N}$ and sparse coefficients $X \in \mathbb{R}^{N \times M}$ via

$$\min_{X,D} \|Y - DX\|_F^2 \qquad \text{s.t. } |X_{:,i}|_0 \leq s \quad \forall i,$$

where the Frobinius norm of a matrix A is $\|A\|_F = (\sum_{i,j}(A_{i,j})^2)^{1/2}$.

This problem is nonconvex, nonsmooth, aend the solution is not unique. How can we deal with these difficulties?

Alterating minimization

Try to tackle

$$\min_{X,D} \|Y - DX\|_F^2 \qquad \text{s.t. } |X_{:,i}|_0 \leq s \quad \forall i,$$

directly using an alternating minimization strategy:

Step 1: For a fixed dictionary D^k use a greedy algorithm to find a good column wise s-sparse coefficient matrix X.

Most common: Orthogonal matching pursuit (OMP). [6] Iteratively increase the support of the solution based on the question which dictionary atom is best suited for decreasing the residual.

First note that the minimization in X decouples column-wise:

$$\|Y - DX\|_F^2 = \sum_i \|Y_{:,i} - DX_{:,i}\|_2^2.$$

[6] See "Orthogonal least squares methods and their application to non-linear system identification", 1989, or "Orthogonal Matching Pursuit : recursive function approximation with application to wavelet decomposition", 1993.

OMP

Since the problem decouples we only need to understand how to approximately solve

$$\min_x \|y - Dx\|_2^2 \qquad \text{s.t. } |x|_0 \leq s$$

OMP algorithm: Assume $\|D_{:,i}\|_2 = 1$ for all i.

Initialize $r^0 = y$, $x^0 = 0$, $I^0 = \emptyset$ and repeat s times

1. Determine $I^k = I^{k-1} \cup \{i\}$ for an index i for which
$$|D_{:,i}^T r^{k-1}| = \|D^T r^{k-1}\|_\infty.$$
2. Compute coefficents
$$x^k = P_{I^k} \operatorname*{argmin}_x \|DP_{I^k} x - y\|^2,$$
where P_{I^k} is the projection onto the support set I^k.
3. Update the residuum
$$r^k = y - Dx^k.$$

OMP

Some remarks:

- The index that is added in each iteration is the one that would be nonzero when solving

$$\|Dx - r^{k-1}\|_2^2 \qquad \text{s.t. } |x|_0 = 1.$$

- For a given support set, the coefficents x^k are the optimal ones.
- The OMP algorithm will not solve the nonconvex sparse recovery problem exactly in general. The latter is an NP-hard problem.
- There exist interesting theoretical studies in which case the OMP algorithm actually solves the nonconvex problem, see for instance J. Tropp, “Greed is Good: Algorithmic Results for. Sparse Approximation”, 2003.

Sparse representation

Back to our problem to tackle

$$\min_{X,D} \|Y - DX\|_F^2 \qquad \text{s.t. } |X_{:,i}|_0 \leq s \quad \forall i,$$

directly.

We already identified a possible step 1: For a fixed dictionary D^k use the OMP algorithm to find a good column wise s-sparse coefficient matrix X.

Step 2: Jointly update the dictionary and coefficients using rank-1 updates by noting that one can write

$$DX = \sum_i D_{:,i} X_{i,:},$$

which is a sum of rank-1 matrices.

This strategy was proposed by Aharon, Elad, Bruckstein. "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation", 2006.

Rank-1 updates

Note that

$$\|Y - DX\|_F^2 = \|Y - \sum_i D_{:,i} X_{i,:}\|_F^2 = \|Y - \sum_{i \neq j} D_{:,i} X_{i,:} - D_{:,j} X_{j,:}\|_F^2.$$

Idea of the algorithm: Define $E^j = Y - \sum_{i \neq j} D_{:,i} X_{i,:}$ and optimize for $D_{:,j}$ and $X_{j,:}$ in

$$\|E^j - D_{:,j} X_{j,:}\|_F^2$$

jointly.

Tempting: Find the best rank-1 approximation by using the SVD $E^j = U \Sigma V^T$, and assign $D_{:,j} = U_{:,1}$, $X_{j,:} = \lambda_1 V_{:,1}$.

Problem: The update $X_{j,:} = \lambda_1 V_{:,1}$ will destroy the sparsity pattern of $X_{j,:}$!

Sparse rank-1 updates

Back to updating $D_{:,j}$ and $X_{j,:}$ based on

$$\|E^j - D_{:,j}X_{j,:}\|_F^2, \tag{1}$$

under the constraint of not changing the sparsity pattern of $X_{j,:}$!

Define $I^j = \{i \mid X_{j,i} \neq 0\}$ and use the MATLAB notation X_{j,I^j} to denote the $1 \times |I^j|$ vector that consists only of those entries that are in I^j.

Replace (1) by

$$\|E^j_{:,I^j} - D_{:,j}X_{j,I^j}\|_F^2.$$

The restriction of $X_{j,:}$ to X_{j,I^j} means that elements that were zero before are kept at zero.

The restriction of E^j to $E^j_{:,I^j}$ means that only those parts of the error E^j matter that were using the j-the dictionary item before.

Sparse rank-1 updates

Codebook Update: Given initial estimates of the dictionary D and the coefficient matrix Y proceed as follows:

For $j \in \{1, ..., N\}$, do

1. Determine $I^j = \{i \mid X_{j,i} \neq 0\}$.
2. Compute $E^j = Y - \sum_{i \neq j} D_{:,i} X_{i,:}$.
3. Compute the SVD of $E^j_{:,I^j} = U \Sigma V^T$.
4. Update

$$X_{j,I^k} \leftarrow \Sigma_{1,1} V_{:,1}$$
$$D_{:,j} \leftarrow U_{:,1}$$

The complete K-SVD algorithm

Determine an initial dictionary D, e.g. by concatenating a DCT basis, a wavelet basis, and some random patches of the training set. Make sure D has normalized columns and iterate:

1. Run the **OMP algorithm** to find coefficients X
2. Run the **Codebook Update** to jointly update D and X

Variant for image denoising: Do not run OMP for a fixed number of iterations s, but rather until $\|Dx - y\|_2 \leq c(\sigma)$, where σ is the noise-level.

Application to inverse problems in imaging:

$$\min_{u,D,X} \lambda\|Au - f\|_2^2 + \|Pu - DX\|_2^2 \qquad \text{s.t. } \|X_{:,i}\|_0 \leq s \ \ \forall i,$$

where P is a linear operator extracting all (possibly overlapping) $\sqrt{n} \times \sqrt{n}$ patches from its input image.

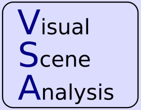
Also see “Sparse Representation for Color Image Restoration” by Mairal, Elad, and Sapiro for extensions to color imaging.

Large inpainting domains

Back to our general inpainting problem. What happens for even larger inpainting domains?

Large inpainting domains

Large inpainting domains



Large inpainting domains

Inpainting based on self-similarity

Interesting approach from Arias et al., *A Variational Framework for Exemplar-Based Image Inpainting*, 2010. Among other suggestions: Extend Poisson image editing into a non-local variational framework.

$$\min_{u,m} \int_{M^c} \int_M m(x,y)(\nabla u(x) - \nabla f(y))^2 \, dx \, dy + R(m),$$

where m is a function containing the non-local weights, f is the input image, M is the inpainting domain, and M^c its complement.

Difficulty: Jointly minimizing for m (with $\int_{M^c} m(x,y)dy = 1$, $m(x,y) \geq 0$) and u.

Inpainting based on self-similarity

Many details to figure out to make it work:

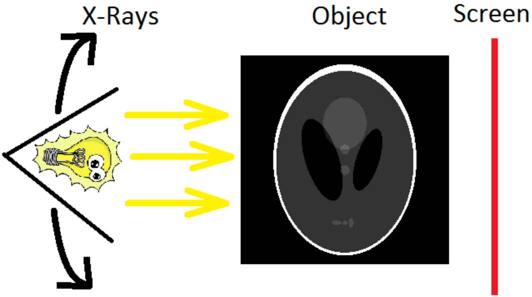
- Determine weights patch-based ($m \leftrightarrow w$ in the paper)
- Introduce data fidelity weights that decrease with the distance to known regions.
- Restrict the number of non-zeros of m, e.g. in the extreme case, pick only one similar patch per unknown pixel.
- Use a coarse-to-fine scheme.

Go through the corresponding paper for details. First (non-variational) approach Criminisi et al., *Region filling and object removal by exemplar-based image inpainting*, 2004.

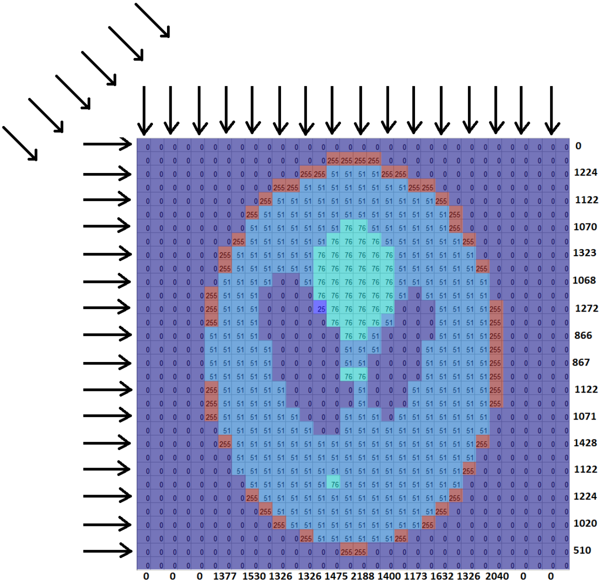
Show toy GUI for image (greedy) patch based inpainting from `https://sourceforge.net/projects/imageinpainting/?source=typ_redirect`.

Tomographic X-ray reconstruction

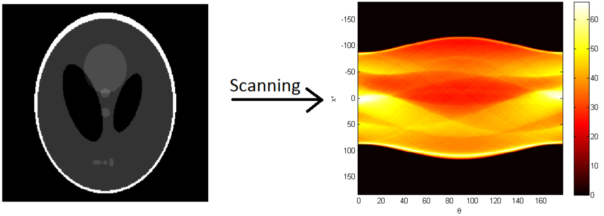
General idea of a computed tomography scanner



General idea of a computed tomography scanner



General idea of a computed tomography scanner



General idea of a computed tomography scanner

What can we reconstruct and how?

The intensity I of an X-ray beam is weakened or *attenuated* depending on the material it travels trough.

The change of intensity is proportional to the intensity I itself, where the proportionality factor is called *attenuation coefficient*.

Since the material changes in space, so does the attenuation coefficient, such that we can describe it as a function $u : \mathbb{R}^2 \to \mathbb{R}$.

An X-ray which travels along a line

$$l(s, \alpha) = \{x \mid \langle \theta(\alpha), x \rangle = s\}$$

for $\theta(\alpha) = (\cos(\alpha), \sin(\alpha))^T$ changes its intensity to

$$I^d = \exp\left(-\int_{l(s,\alpha)} u(x)\ dx^{\perp} \right)\ I^0$$

where we assume u to be zero outside of our region of interest.

Radon transform

We define the *Radon transform* of a function $u : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$ as

$$\begin{aligned}\mathcal{R}(u)(s,\alpha) &= \int_{l(s,\alpha)} u(x)\; dx^\perp \\ &= \int_{-\infty}^{\infty} u(s\theta(\alpha) + t\theta(\alpha)^\perp)\; dt \\ &= \int_{-\infty}^{\infty} u(s\cos(\alpha) - t\sin(\alpha), s\sin(\alpha) + t\cos(\alpha))\; dt\end{aligned}$$

Based on our formula for the change of intensity of an X-ray, if we send out a ray with known intensity I^0 and measure its intensity I^d behind an object, we can determine

$$\log(I^0) - \log(I^d) = \mathcal{R}(u)(s,\alpha).$$

Reconstruction: Find u from measurements $\mathcal{R}(u)(s,\alpha)$!

Radon transform

From the linearity of the integral, it immediately follows that taking the Radon transform $\mathcal{R}$ is a linear operator!

Structurally, the reconstruction task does not differ from any of the previous examples!

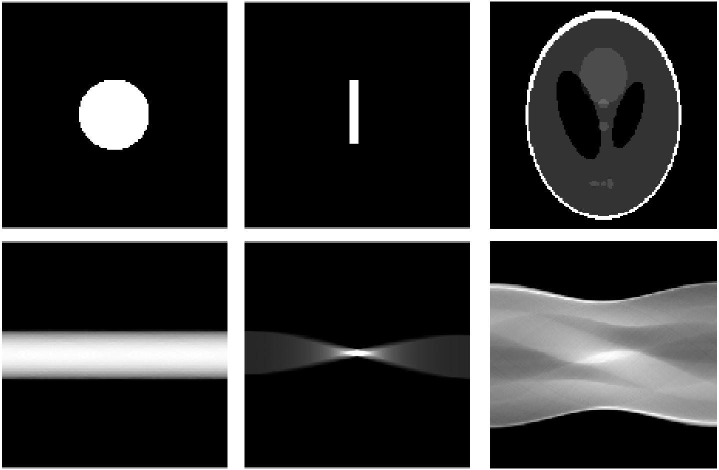
In the continuous setting, there exists a closed-form inversion formula for reconstructing u from $\mathcal{R}(u)$ as you will prove in the exercise.

The standard reconstruction technique called *filtered-backprojection* is a modified version of the inversion formula to stabilize the reconstruction.

Different from the previous examples: The measured data is not image-like!

Radon transform

Some examples of images and their *sinograms*.



Radon and Fourier transforms

The Radon transform is closely related to the Fourier transform.

This allows to derive an inversion formula for the radon transformation in the continuous setting.

Since the inversion is ill-posed, one rather uses a technique called *filtered-backprojection*. Try *help iradon* in MATLAB for some examples.

However, since line integrals are linear operators, we should also be able to derive a variational method for the problem!

Discretization of the Radon transform

How can we implement the transform numerically?

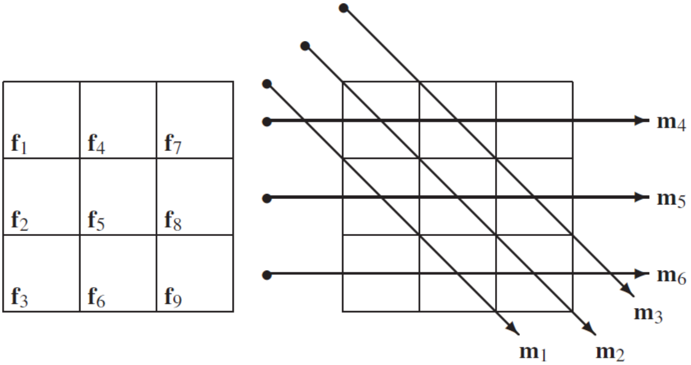
Simplest way: Use a pixel grid and assume the absorption coefficient to be constant within each grid cell.

Remaining question: What is the corresponding approximation of the line integral for such a discretization?

Weighted sum over the pixels that the ray intersects with! The weights is the distance the ray travels in the pixel.

Discretization of the Radon transform

From: **Jenifer L. Mueller and Samuli Siltanen, “Linear and Nonlinear Inverse Problems with Practical Applications”.**



$$R = \begin{pmatrix} 0 & \sqrt{2} & 0 & 0 & 0 & \sqrt{2} & 0 & 0 & 0 \\ \sqrt{2} & 0 & 0 & 0 & \sqrt{2} & 0 & 0 & 0 & \sqrt{2} \\ 0 & 0 & 0 & \sqrt{2} & 0 & 0 & 0 & \sqrt{2} & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Computational considerations

Besides the Radon transform being an example in which the result $\mathcal{R}(u)$ of applying a linear operator does not yield an image anymore, it also yields an example in which it can quickly become too expensive to store a matrix representation of $\mathcal{R}$.

We therefore need fast methods to apply the Radon transform and its adjoint which are not based on a matrix representation of the operator $\mathcal{R}$.

We will practice dealing with such a situation in an inefficient but conceptually important way in MATLAB using function handles for `radon` and `iradon` in the exercises.