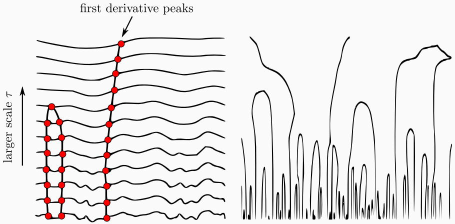
ECE 285

Image and video restoration

Chapter IV – Variational methods

Charles Deledalle

June 9, 2019



Heat equation

Heat diffusion – Motivation

$\xrightarrow{?}$

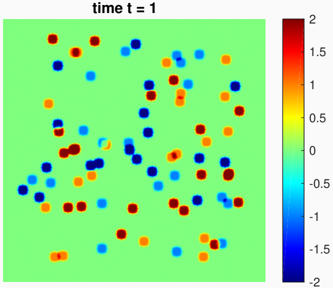
- How can we remove noise from an image?
- What image can best explain this noisy observation?
- Takes inspiration from our physical world.

Best explanation is the one with maximal entropy.

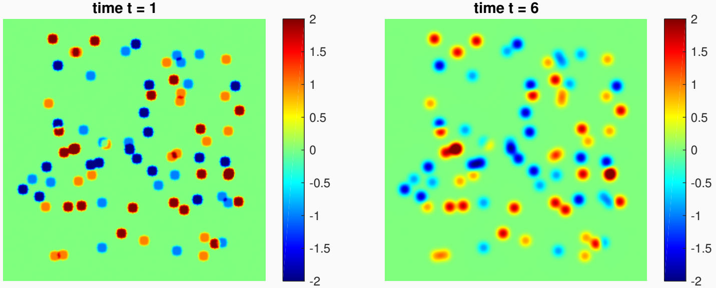
- Heat, in an isolated system, evolves such that

  the total entropy increases over time.

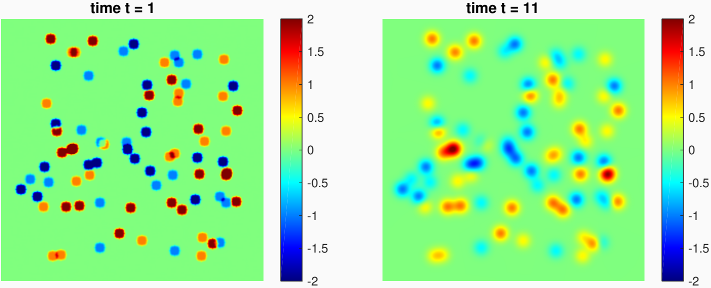
Heat diffusion



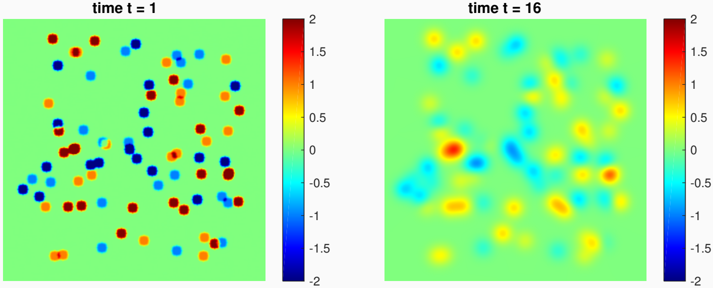
Heat diffusion



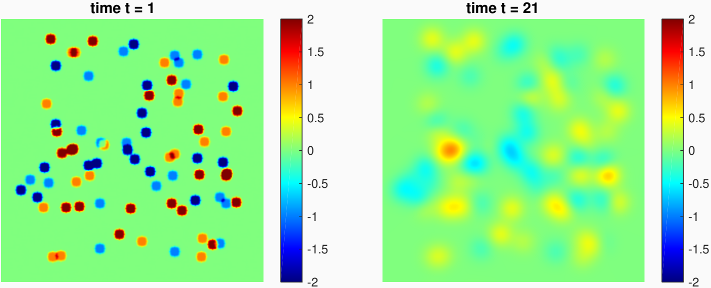
Heat diffusion



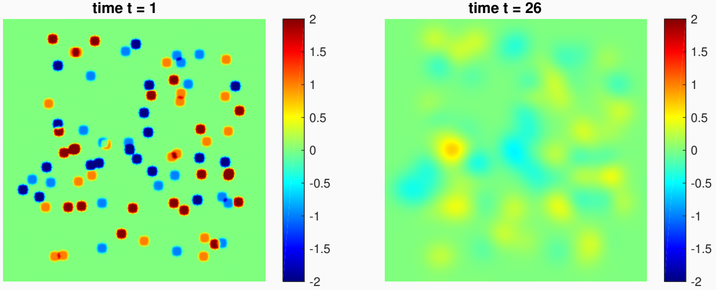
Heat diffusion



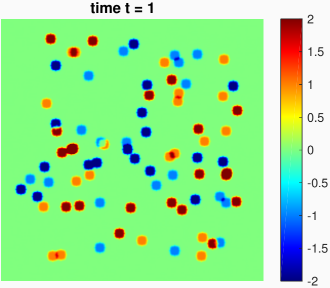
Heat diffusion

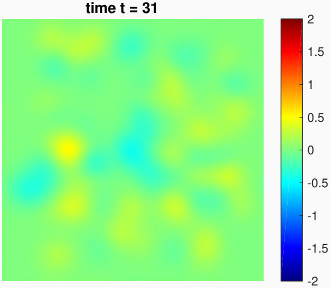


Heat diffusion

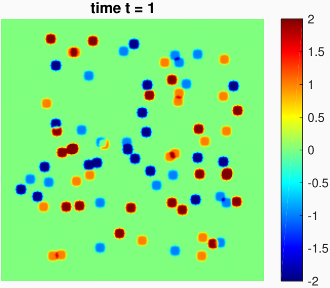


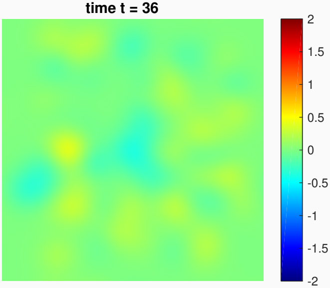
Heat diffusion



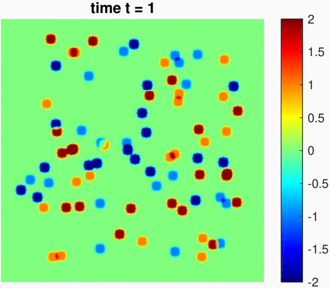


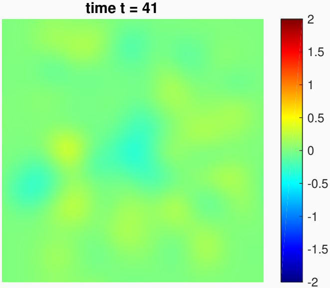
Heat diffusion



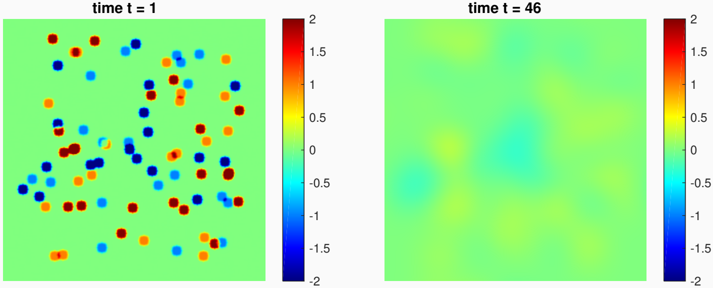


Heat diffusion

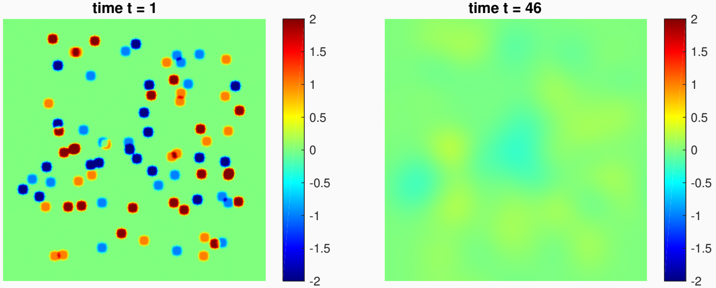




Heat diffusion



Heat diffusion



Heat diffusion acts as a denoiser

- Spatial fluctuations of temperatures vanish with time (maximum entropy),
- Think of pixel values as temperature,
- Can heat diffusion help us to reduce noise?

Heat equation

The heat equation, a Partial Differential Equation (PDE), given by

$$\frac{\partial x}{\partial t}(s,t) = \alpha \Delta x(s,t) \quad \text{or in short} \quad \frac{\partial x}{\partial t} = \alpha \Delta x \quad \text{and} \quad x(s,0) = y(s)$$

$+$ some boundary conditions and where

- $s = (s_1, s_2) \in [0,1]^2$: space location
- $t \geqslant 0$: time location
- $x(s,t) \in \mathbb{R}$: temperature at position s and time t
- $\alpha > 0$: thermal conductivity constant
- Δ: Laplacian operator

$$\Delta = \frac{\partial^2}{\partial s_1^2} + \frac{\partial^2}{\partial s_2^2}$$

The rate of change is proportional to the spatial curvature of the temperature.

How to solve the heat equation?

2 solutions:

1. Heat equation $\longrightarrow$ Discrete equation $\longrightarrow$ Numerical scheme

2. Heat equation $\longrightarrow$ Continuous solution $\longrightarrow$ Discretization

How to solve the heat equation?

2 solutions:

1. Heat equation $\longrightarrow$ Discrete equation $\longrightarrow$ Numerical scheme

2. Heat equation $\longrightarrow$ Continuous solution $\longrightarrow$ Discretization

Discretization of the working space

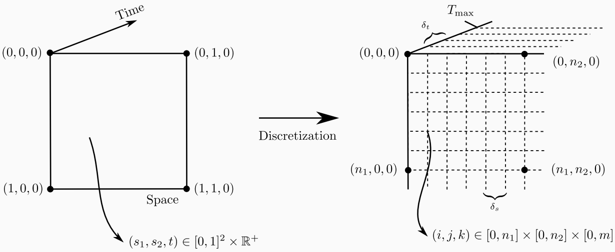
- Periodical boundary conditions

$$x(0, s_2, t) = x(1, s_2, t) \quad \text{and} \quad x(s_1, 0, t) = x(s_1, 1, t).$$

- Map the discrete grid to the continuous coordinates (s_1, s_2, t)

$$(s_1, s_2, t) = (i\delta_{s_1}, j\delta_{s_2}, k\delta_t)$$

where $(i, j) \in [0, n_1] \times [0, n_2]$, $k \in [0, m]$, $\delta_{s_i} = \dfrac{1}{n_i}$ and $\delta_t = \dfrac{T_{\max}}{m}$.



Heat equation – Discretization

- Then, replace function x by its discrete version:

$$x_{i,j}^{k} = x(i\delta_{s_1}, j\delta_{s_2}, k\delta_t)$$

- i: index for pixels with first coordinate $s_1 = i\delta_{s_1}$
- j: index for pixels with second coordinate $s_2 = j\delta_{s_2}$
- k: is an index for time $t = k\delta_t$

⚠ **The notation x^k is not “x to the power k” but “x at time index k”.**

Heat equation – Finite differences

Recall: we want to discretize

$$\frac{\partial x}{\partial t}(s,t) = \alpha \Delta x(s,t) \quad \text{and} \quad x(s,0) = y(s)$$

Finite differences

- Replace first order derivative by forward finite difference in time

$$\frac{\partial x}{\partial t}(i\delta_{s_1}, j\delta_{s_2}, k\delta_t) \approx \frac{x_{i,j}^{k+1} - x_{i,j}^{k}}{\delta_t}$$

Heat equation – Finite differences

Recall: we want to discretize

$$\frac{\partial x}{\partial t}(s,t) = \alpha \Delta x(s,t) \quad \text{and} \quad x(s,0) = y(s)$$

Finite differences

- Replace first order derivative by forward finite difference in time

$$\frac{\partial x}{\partial t}(i\delta_{s_1}, j\delta_{s_2}, k\delta_t) \approx \frac{x_{i,j}^{k+1} - x_{i,j}^{k}}{\delta_t}$$

- Replace second order derivative by central finite difference in space

$$\Delta x(i\delta_{s_1}, j\delta_{s_2}, k\delta_t) \approx \frac{x_{i-1,j}^{k} + x_{i+1,j}^{k} + x_{i,j-1}^{k} + x_{i,j-1}^{k} - 4x_{i,j}^{k}}{\delta_{s_1}\delta_{s_2}}$$

Heat equation – Finite differences

Recall: we want to discretize

$$\frac{\partial x}{\partial t}(s,t) = \alpha \Delta x(s,t) \quad \text{and} \quad x(s,0) = y(s)$$

Finite differences

- Rewrite everything in matrix/vector form

$$\frac{\partial x}{\partial t}(\cdot,\cdot,k\delta_t) \approx \frac{1}{\delta_t}(x^{k+1} - x^k) \quad \text{and} \quad \Delta x(\cdot,\cdot,k\delta_t) \approx \frac{1}{\delta_{s_1}\delta_{s_2}} \Delta x^k$$

  where Δ in the right-hand side is the discrete Laplacian.
- We get

$$\frac{1}{\delta_t}(x^{k+1} - x^k) = \frac{\alpha}{\delta_{s_1}\delta_{s_2}} \Delta x^k \quad \text{and} \quad x^0 = y$$

Heat equation – Discrete Laplacian

$$
\Delta x =
\begin{array}{c}
\begin{array}{ccccc|ccccc|ccccc|ccccc}
 & & \text{First line} & & & & & \text{Second line} & & & & & \cdots & & & & & \text{Last line} & & \\
\end{array}\\
\left(
\begin{array}{ccccc|ccccc|ccccc|ccccc}
4 & -1 & & & -1 & -1 & & & & & & & & & & -1 & & & & \\
-1 & 4 & -1 & & & & -1 & & & & & & & & & & -1 & & & \\
 & \ddots & \ddots & \ddots & & & & \ddots & & & & & 0 & & & & & \ddots & & \\
 & & \ddots & \ddots & \ddots & & & & \ddots & & & & & & & & & & \ddots & \\
-1 & & & -1 & 4 & & & & & -1 & & & & & & & & & & -1 \\
\hline
-1 & & & & & 4 & -1 & & & -1 & -1 & & & & & & & & & \\
 & -1 & & & & -1 & 4 & -1 & & & & -1 & & & & & & & & \\
 & & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & & & & & 0 & & \\
 & & & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & & & & & & \\
 & & & & -1 & -1 & & & -1 & 4 & & & & & -1 & & & & & \\
\hline
 & & & & & & & & & & & & & & & & & & & \\
 & & 0 & & & & & \ddots & & & & & \ddots & & & & & \ddots & & \\
 & & & & & & & & & & & & & & & & & & & \\
\hline
-1 & & & & & & & & & & -1 & & & & & 4 & -1 & & & -1 \\
 & -1 & & & & & & & & & & -1 & & & & -1 & 4 & -1 & & \\
 & & \ddots & & & & & 0 & & & & & \ddots & & & & \ddots & \ddots & \ddots & \\
 & & & \ddots & & & & & & & & & & \ddots & & & & \ddots & \ddots & \ddots \\
 & & & & -1 & & & & & & & & & & -1 & -1 & & & -1 & 4
\end{array}
\right)
\end{array}
\begin{pmatrix}
x_{1,1} \\ x_{1,2} \\ \vdots \\ \vdots \\ x_{1,n_2} \\ \hline
x_{2,1} \\ x_{2,2} \\ \vdots \\ \vdots \\ x_{2,n_2} \\ \hline
\\ \vdots \\ \\ \hline
x_{n_1,1} \\ x_{n_1,2} \\ \vdots \\ \vdots \\ x_{n_1,n_2}
\end{pmatrix}
$$

because of periodical boundary conditions.

Forward discretized scheme – Explicit Euler scheme

The heat equation $\dfrac{\partial x}{\partial t} = \alpha \Delta x$ and $x(s, 0) = y(s)$

rewrites as $\dfrac{1}{\delta_t}(x^{k+1} - x^k) = \dfrac{\alpha}{\delta_{s_1}\delta_{s_2}}\Delta x^k$ and $x^0 = y$

which leads to the iterative scheme, that repeats for $k = 0$ to m

$$x^{k+1} = x^k + \gamma \Delta x^k \quad \text{and} \quad x^0 = y \quad \text{where} \quad \gamma = \frac{\alpha \delta_t}{\delta_{s_1}\delta_{s_2}}$$

Heat equation – Explicit Euler scheme

Forward discretized scheme – Explicit Euler scheme

The heat equation $\dfrac{\partial x}{\partial t} = \alpha \Delta x$ and $x(s, 0) = y(s)$

rewrites as $\dfrac{1}{\delta_t}(x^{k+1} - x^k) = \dfrac{\alpha}{\delta_{s_1}\delta_{s_2}} \Delta x^k$ and $x^0 = y$

which leads to the iterative scheme, that repeats for $k = 0$ to m

$$x^{k+1} = x^k + \gamma \Delta x^k \quad \text{and} \quad x^0 = y \quad \text{where} \quad \gamma = \frac{\alpha \delta_t}{\delta_{s_1}\delta_{s_2}}$$

Convergence: $|x^k_{i,j} - x(i\delta_{s_1}, j\delta_{s_2}, k\delta_t)| \xrightarrow[\substack{\delta_{s_1}\to 0\\ \delta_{s_2}\to 0\\ \delta_t \to 0}]{} 0,$ for all (i, j, k)

Heat equation – Explicit Euler scheme

Forward discretized scheme – Explicit Euler scheme

The heat equation $\dfrac{\partial x}{\partial t} = \alpha \Delta x$ and $x(s, 0) = y(s)$

rewrites as $\dfrac{1}{\delta_t}(x^{k+1} - x^k) = \dfrac{\alpha}{\delta_{s_1}\delta_{s_2}} \Delta x^k$ and $x^0 = y$

which leads to the iterative scheme, that repeats for $k = 0$ to m

$$x^{k+1} = x^k + \gamma \Delta x^k \quad \text{and} \quad x^0 = y \quad \text{where} \quad \gamma = \frac{\alpha \delta_t}{\delta_{s_1}\delta_{s_2}}$$

Convergence: $|x^k_{i,j} - x(i\delta_{s_1}, j\delta_{s_2}, k\delta_t)| \xrightarrow[\substack{\delta_{s_1} \to 0 \\ \delta_{s_2} \to 0 \\ \delta_t \to 0}]{} 0,$ for all (i, j, k)

δ_{s_1} and δ_{s_2} are fixed (by the size of the image grid).

δ_t influences the number of iterations k used to reach $t = k\delta_t$.

δ_t should be small enough (for convergence),
and large enough (for computation time).

Heat equation – Explicit Euler scheme

Stability

- The discretization scheme is stable, if there exists $C > 0$ such that

$$\text{for all } (i, j, k), \quad |x_{i,j}^k| \leqslant C|y_{i,j}|.$$

- Stability prevents the iterates from diverging.
- If moreover numerical errors do not accumulate, x^k converges with k.

Heat equation – Explicit Euler scheme

Stability

- The discretization scheme is stable, if there exists $C > 0$ such that

$$\text{for all } (i,j,k), \quad |x^k_{i,j}| \leqslant C|y_{i,j}|.$$

- Stability prevents the iterates from diverging.
- If moreover numerical errors do not accumulate, x^k converges with k.

Courant-Friedrichs-Lewy (CFL) conditions

The sequence x_k is stable if: $\gamma = \dfrac{\alpha\delta_t}{\delta_{s_1}\delta_{s_2}} < \dfrac{1}{2d}$ where $d = 2$ for images

In particular we get $m > 2d\alpha T_{\max} n_1 n_2$

#iterations increases linearly with #pixels

$\Rightarrow$ for k to reach m, at least $O(n_1^2 n_2^2)$ operations, *i.e.*, it is really slow. ☹

Heat equation – Explicit Euler scheme

Geometric progression

The explicit Euler scheme can be rewritten as

$$x^{k+1} = x^k + \gamma \Delta x^k = (\mathrm{Id}_n + \gamma \Delta) x^k, \quad (n = n_1 n_2)$$

it is a geometric progression, hence: $x^k = (\mathrm{Id}_n + \gamma \Delta)^k y$

Heat equation – Explicit Euler scheme

Geometric progression

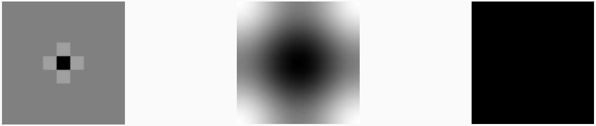
The explicit Euler scheme can be rewritten as

$$x^{k+1} = x^k + \gamma \Delta x^k = (\mathrm{Id}_n + \gamma \Delta) x^k, \quad (n = n_1 n_2)$$

it is a geometric progression, hence: $x^k = (\mathrm{Id}_n + \gamma\Delta)^k y$

Diagonalization

- Δ performs a periodical convolution with kernel: $\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$
- Diagonal in the discrete Fourier domain: $\Delta = \boldsymbol{F}^{-1} \boldsymbol{\Lambda} \boldsymbol{F}$, with $\boldsymbol{\Lambda}$ diagonal



(a) Δ (b) Re$[\mathcal{F}(\Delta)]$ (c) Im$[\mathcal{F}(\Delta)]$

Heat equation – Explicit Euler scheme

Geometric progression + Diagonalization

- The explicit Euler scheme becomes

$$
\begin{aligned}
x^k &= (\mathrm{Id}_n + \gamma \boldsymbol{F}^{-1} \boldsymbol{\Lambda} \boldsymbol{F})^k y \\
&= (\boldsymbol{F}^{-1}\boldsymbol{F} + \gamma \boldsymbol{F}^{-1} \boldsymbol{\Lambda} \boldsymbol{F})^k y \\
&= (\boldsymbol{F}^{-1}(\mathrm{Id} + \gamma \boldsymbol{\Lambda})\boldsymbol{F})^k y \\
&= \underbrace{\boldsymbol{F}^{-1}(\mathrm{Id} + \gamma \boldsymbol{\Lambda})\boldsymbol{F} \times \boldsymbol{F}^{-1}(\mathrm{Id} + \gamma \boldsymbol{\Lambda})\boldsymbol{F} \times \ldots \times \boldsymbol{F}^{-1}(\mathrm{Id} + \gamma \boldsymbol{\Lambda})\boldsymbol{F}}_{k \text{ times}} \, y \\
&= \boldsymbol{F}^{-1} \underbrace{(\mathrm{Id} + \gamma \boldsymbol{\Lambda}) \times \ldots \times (\mathrm{Id} + \gamma \boldsymbol{\Lambda})}_{k \text{ times}} \boldsymbol{F} y \\
&= \boldsymbol{F}^{-1} \underbrace{(\mathrm{Id} + \gamma \boldsymbol{\Lambda})^k}_{\text{diagonal matrix}} \boldsymbol{F} y
\end{aligned}
$$

- The explicit Euler solution is a convolution,
- Solution in $O(n \log n)$ whatever k. ☺

Heat equation – Explicit Euler scheme – Results

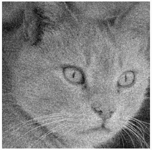
```python
# Load image (assumed to be square)
x = plt.imread('assets/cat.png')
n1, n2 = x.shape
sig = 20/255
y = x + sig * np.random.randn(n1, n2)

# Create Laplacian kernel in Fourier
nu = (im.kernel('laplacian1'),
      im.kernel('laplacian2'))
L = im.kernel2fft(nu, n1, n2, separable='sum')

# Define problem setting (T = m * dt)
T     = 1e-4
alpha = 1
rho   = .99
ds2   = 1 / (n1 * n2)
dt    = rho * ds2 / (4 * alpha)
gamma = alpha * dt / ds2
m     = np.round(T / dt)

# Compute explicit Euler solution
K_ee  = (1 + gamma * L)**m
x_ee = im.convolvefft(y, K_ee)
```

CFL condition: $\gamma = \frac{\alpha \delta_t}{\delta_s^2} < \frac{1}{4}$

$$\Rightarrow \delta_t < \frac{\delta_s^2}{4\alpha}$$

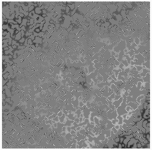
$$\Rightarrow \delta_t = \rho \frac{\delta_s^2}{4\alpha} \quad \text{with} \quad \rho < 1$$

(a) x (unknown)



(b) y (observation)

(c) $T{=}10^{-4}$, $\rho{=}0.99$



(d) $T{=}10^{-4}$, $\rho{=}1.30$

Heat equation – Implicit Euler scheme

Backward discretized scheme – Implicit Euler scheme

If instead we choose a backward difference in time

$$\frac{1}{\delta_t}(x^{k+1} - x^k) = \frac{\alpha}{\delta_{s_1}\delta_{s_2}}\Delta x^{k+1} \quad \text{and} \quad x^0 = y$$

this leads to the iterative scheme

$$x^{k+1} = (\mathrm{Id}_n - \gamma\Delta)^{-1}x^k \quad \text{and} \quad x^0 = y.$$

This sequence is stable whatever γ, but requires solving a linear system. ☹

Geometric progression and diagonalization

- Geometric progression: $x^k = (\mathrm{Id}_n - \gamma\Delta)^{-k} y$
- Again, since $\Delta = \boldsymbol{F}^{-1}\boldsymbol{\Lambda}\boldsymbol{F}$ is diagonal in the Fourier domain

$$x^k = \boldsymbol{F}^{-1}(\mathrm{Id}_n - \gamma\boldsymbol{\Lambda})^{-k}\boldsymbol{F}y.$$

- The implicit Euler solution is again a convolution.
- Can be computed in $O(n \log n)$ whatever k. ☺

Heat equation – Implicit Euler scheme

```python
# Compute explicit Euler solution
K_ee = (1 + gamma * L)**k
x_ee = im.convolvefft(y, K_ee)
```

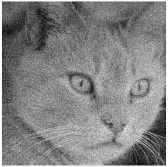
```python
# Compute implicit Euler solution
K_ie = 1 / (1 - gamma * L)**k
x_ie = im.convolvefft(y, K_ie)
```

Heat equation – Implicit Euler scheme

```
# Compute explicit Euler solution
K_ee = (1 + gamma * L)**k
x_ee = im.convolvefft(y, K_ee)
```

```
# Compute implicit Euler solution
K_ie = 1 / (1 - gamma * L)**k
x_ie = im.convolvefft(y, K_ie)
```

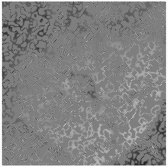
(a) x (unknown)



(b) y (observation)

Explicit Euler

(c) $T = 10^{-4}$, $\rho = 0.99$



(d) $T = 10^{-4}$, $\rho = 1.30$

Implicit Euler

(e) $T = 10^{-4}$, $\rho = 0.99$

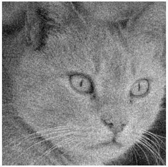
(f) $T = 10^{-4}$, $\rho = 1.30$

Heat equation – Implicit Euler scheme

```
# Compute explicit Euler solution
K_ee = (1 + gamma * L)**k
x_ee = im.convolvefft(y, K_ee)
```

```
# Compute implicit Euler solution
K_ie = 1 / (1 - gamma * L)**k
x_ie = im.convolvefft(y, K_ie)
```

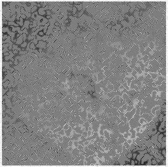
(a) x (unknown)



(b) y (observation)

Explicit Euler

(c) $T{=}10^{-4}$, $\rho{=}0.99$



(d) $T{=}10^{-4}$, $\rho{=}1.30$

Implicit Euler

(e) $T{=}10^{-4}$, $\rho{=}0.99$

(f) $T{=}10^{-4}$, $\rho{=}1.30$

Q: How both schemes compare to the continuous solution when $\rho < 1$?

How to solve the heat equation?

2 solutions:

1. Heat equation $\longrightarrow$ Discrete equation $\longrightarrow$ Numerical scheme

2. Heat equation $\longrightarrow$ Continuous solution $\longrightarrow$ Discretization

Heat equation – Continuous solution

Theorem

- *Consider the continuous heat equation defined as*

$$\frac{\partial x}{\partial t}(s, t) = \alpha \Delta x(s, t) \quad \textit{and} \quad x(s, 0) = y(s)$$

where $s \in \mathbb{R}^d$ (no restrictions to $[0, 1]^d$, without boundary conditions).

- *The exact solution is given by the d-dimensional Gaussian convolution*

$$x(s, t) = (y * \mathcal{G}_{2\alpha t})(s) = \int_{\mathbb{R}^d} y(s - u) \frac{1}{\sqrt{4\pi\alpha t}^d} e^{-\frac{\|u\|_2^2}{4\alpha t}} \, \mathrm{d}u$$

($d = 2$ for images).

- *This is called the fundamental solution of the heat equation.*

Heat equation – Continuous solution

Proof in the 1d case.

- In the 1d case the Heat equation reads as

$$\frac{\partial x}{\partial t} = \alpha \Delta x \overset{1d}{=} \alpha \frac{\partial^2 x}{\partial s^2} \quad \text{and} \quad x(s, 0) = y(s)$$

- Taking the spatial Fourier transform (with respect to s) in both sides gives

$$\mathcal{F}_s \left[\frac{\partial x}{\partial t} \right] = \alpha \mathcal{F}_s \left[\frac{\partial^2 x}{\partial s^2} \right] \quad \text{and} \quad \mathcal{F}_s[x](u, 0) = \mathcal{F}_s[y](u)$$

Heat equation – Continuous solution

Proof in the 1d case.

- In the 1d case the Heat equation reads as

$$\frac{\partial x}{\partial t} = \alpha \Delta x \overset{1d}{=} \alpha \frac{\partial^2 x}{\partial s^2} \quad \text{and} \quad x(s,0) = y(s)$$

- Taking the spatial Fourier transform (with respect to s) in both sides gives

$$\mathcal{F}_s\left[\frac{\partial x}{\partial t}\right] = \alpha \mathcal{F}_s\left[\frac{\partial^2 x}{\partial s^2}\right] \quad \text{and} \quad \mathcal{F}_s[x](u,0) = \mathcal{F}_s[y](u)$$

$$\Rightarrow \quad \frac{\partial \mathcal{F}_s[x]}{\partial t} = -4\pi^2 u^2 \alpha \cdot \mathcal{F}_s[x] \qquad \left(\frac{d^n f(t)}{dt^n} \to (2\pi i u)^n \hat{f}(u)\right)$$

- This is a first order differential equation, $x'(t) = ax(t)$, whose solution is

Heat equation – Continuous solution

Proof in the 1d case.

- In the 1d case the Heat equation reads as

$$\frac{\partial x}{\partial t} = \alpha \Delta x \overset{1d}{=} \alpha \frac{\partial^2 x}{\partial s^2} \quad \text{and} \quad x(s, 0) = y(s)$$

- Taking the spatial Fourier transform (with respect to s) in both sides gives

$$\mathcal{F}_s \left[\frac{\partial x}{\partial t} \right] = \alpha \mathcal{F}_s \left[\frac{\partial^2 x}{\partial s^2} \right] \quad \text{and} \quad \mathcal{F}_s[x](u, 0) = \mathcal{F}_s[y](u)$$

$$\Rightarrow \quad \frac{\partial \mathcal{F}_s[x]}{\partial t} = -4\pi^2 u^2 \alpha \cdot \mathcal{F}_s[x] \qquad \left(\frac{d^n f(t)}{dt^n} \to (2\pi i u)^n \hat{f}(u) \right)$$

- This is a first order differential equation, $x'(t) = ax(t)$, whose solution is

$$\mathcal{F}_s[x](u, t) = \mathcal{F}_s[y](u) \cdot e^{(-4\pi^2 \alpha u^2)t}$$

- Products in Fourier domain corresponds to convolutions in the spatial domain, which concludes the proof since $\mathcal{F}[\mathcal{G}_{\gamma^2}] = \sqrt{2\pi\gamma^2}^d \mathcal{G}_{1/4\pi^2\gamma^2}$

Proof in the 1d case.

- In the 1d case the Heat equation reads as

$$\frac{\partial x}{\partial t} = \alpha \Delta x \overset{1d}{=} \alpha \frac{\partial^2 x}{\partial s^2} \quad \text{and} \quad x(s, 0) = y(s)$$

- Taking the spatial Fourier transform (with respect to s) in both sides gives

$$\mathcal{F}_s \left[\frac{\partial x}{\partial t}\right] = \alpha \mathcal{F}_s \left[\frac{\partial^2 x}{\partial s^2}\right] \quad \text{and} \quad \mathcal{F}_s[x](u, 0) = \mathcal{F}_s[y](u)$$

$$\Rightarrow \quad \frac{\partial \mathcal{F}_s[x]}{\partial t} = -4\pi^2 u^2 \alpha \cdot \mathcal{F}_s[x] \qquad \left(\tfrac{d^n f(t)}{dt^n} \to (2\pi i u)^n \hat{f}(u)\right)$$

- This is a first order differential equation, $x'(t) = ax(t)$, whose solution is

$$\mathcal{F}_s[x](u, t) = \mathcal{F}_s[y](u) \cdot e^{(-4\pi^2 \alpha u^2)t}$$

- Products in Fourier domain corresponds to convolutions in the spatial domain, which concludes the proof since $\mathcal{F}[\mathcal{G}_{\gamma^2}] = \sqrt{2\pi\gamma^2}^d \mathcal{G}_{1/4\pi^2\gamma^2}$

$$\mathcal{F}_s^{-1}\left[e^{-4\pi^2 \alpha t u^2}\right] = \frac{1}{\sqrt{4\pi\alpha t}} e^{-\frac{s^2}{4\alpha t}} = \mathcal{G}_{2\alpha t}(s)$$

□

Heat equation – Discretization of the solution

Continuous solution for $d = 2$

$$x(s_1, s_2, t) = \frac{1}{4\pi\alpha t} \iint_{-\infty}^{+\infty} y(s_1 - u, s_2 - v) e^{-\frac{u^2+v^2}{4\alpha t}} \, \mathrm{d}u \, \mathrm{d}v = (y * \mathcal{G}_{2\alpha t})(s_1, s_2)$$

Discretization

$$x_{i,j}^k = x(i\delta_s, j\delta_s, k\delta_t) = \frac{1}{4\pi\alpha k\delta_t} \iint_{-\infty}^{+\infty} y(i\delta_s - u, j\delta_s - v) e^{-\frac{u^2+v^2}{4\alpha k\delta_t}} \, \mathrm{d}u \, \mathrm{d}v$$

$$= \frac{\delta_s^2}{4\pi\alpha\delta_t k} \iint_{-\infty}^{+\infty} y(i\delta_s - u\delta_s, j\delta_s - v\delta_s) e^{-\frac{\delta_s^2(u^2+v^2)}{4\alpha\delta_t k}} \, \mathrm{d}u \, \mathrm{d}v \qquad \left(\substack{\text{Change of variables:} \\ u \to \delta_s u \text{ and } v \to \delta_s v}\right)$$

$$= \frac{1}{4\pi\gamma k} \iint_{-\infty}^{+\infty} y((i-u)\delta_s, (j-v)\delta_s) e^{-\frac{u^2+v^2}{4\gamma k}} \, \mathrm{d}u \, \mathrm{d}v \qquad \left(\text{Recall: } \gamma = \frac{\alpha\delta_t}{\delta_s^2}\right)$$

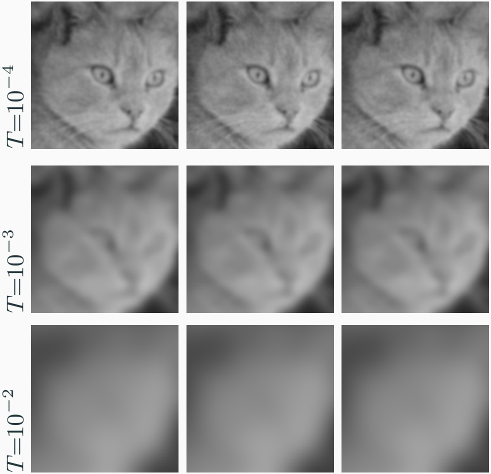
$$\approx \underbrace{\frac{1}{4\pi\gamma k} \sum_{u \in \mathbb{Z}} \sum_{v \in \mathbb{Z}} y_{i-u,j-v} e^{-\frac{u^2+v^2}{4\gamma k}}}_{\text{discrete convolution}} \qquad \text{(Midpoint Riemann sum)}$$

$$= (y * \mathcal{G}_{2\gamma k})_{i,j}$$

```python
# Compute explicit Euler solution
K_ee = (1 + gamma * L)**k
x_ee = im.convolvefft(y, K_ee)

# Compute implicit Euler solution
K_ie = 1 / (1 - gamma * L)**k
x_ie = im.convolvefft(y, K_ie)

# Compute continuous solution
u, v = im.fftgrid(n1, n2)
K_cs = np.exp(-(u**2 + v**2) / (4*gamma*k)) / (4*np.pi*gamma*k)
K_cs = nf.fft2(K_cs, axes=(0, 1))
x_cs = im.convolvefft(y, K_cs)
```

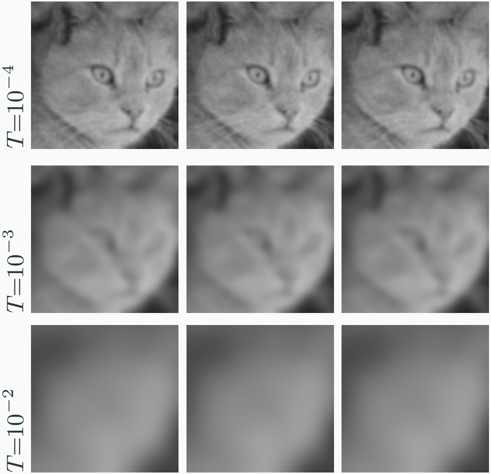
Heat equation – Comparing the results

(a) y (observation)



(b) Explicit Euler (c) Implicit Euler (d) Continuous

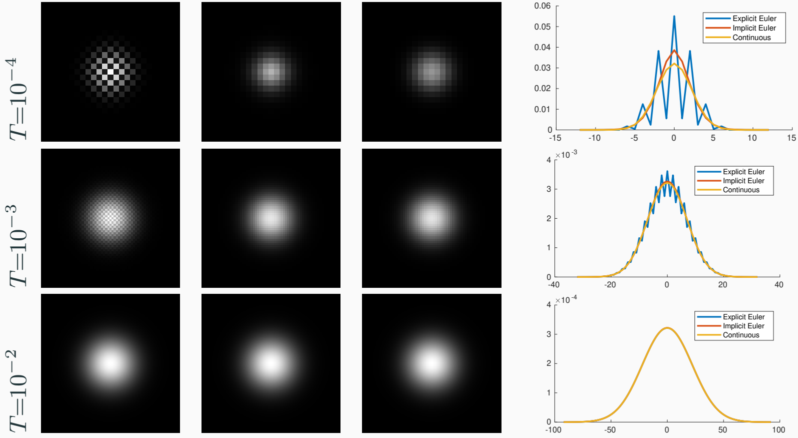
(a) y (observation)



(b) Explicit Euler (c) Implicit Euler (d) Continuous

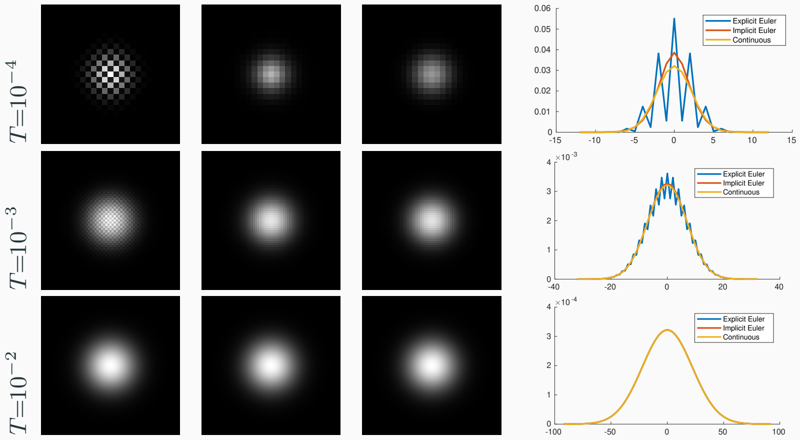
The three schemes provide similar solutions in $O(n \log n)$.

Heat equation – Comparing the convolution kernels



(a) Explicit Euler (b) Implicit Euler (c) Continuous (d) 1D slice

Heat equation – Comparing the convolution kernels



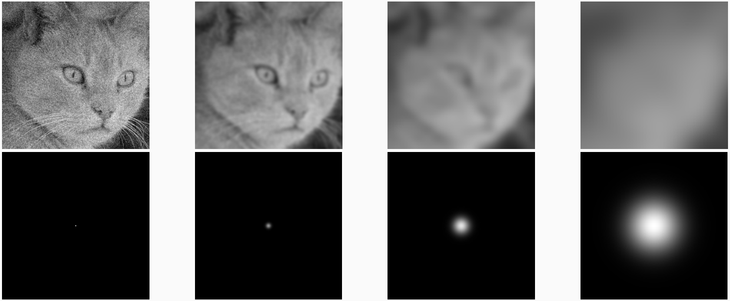
(a) Explicit Euler (b) Implicit Euler (c) Continuous (d) 1D slice

For the same choice of δ_t satisfying the CFL condition, the implicit and continuous solutions have less oscillations. All three converge with t to the same solution.

Heat equation – Summary

Summary

- Solutions of the heat equations reduce fluctuations/details of the image,
- The continuous solution is a Gaussian convolution (LTI filter),
- Discretizations lead to near Gaussian convolutions,
- The width of the convolution kernel increases with time t,
- For $t \to \infty$, the solution is the constant mean image.



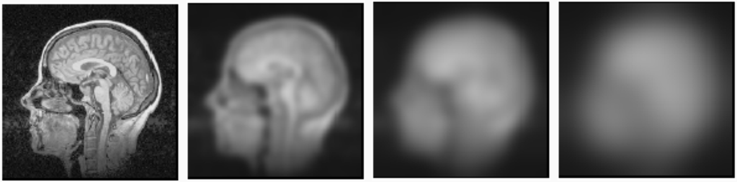
(a) $t = 0$ (b) $t = 10^{-4}$ (c) $t = 10^{-3}$ (d) $t = 10^{-2}$

Scale space

Scale space

Definition (Scale space)

- A family of images $x(s_1, s_2, t)$, where
  - t is the scale-space parameter
  - $x(s_1, s_2, 0) = y(s_1, s_2)$ is the original image
  - increasing t corresponds to coarser resolutions
- and satisfying (scale-space conditions)
  - causality: coarse details are “caused” by fine details
  - new details should not arise in coarse scale images



Gaussian blurring is a local averaging operation.
It does not respect natural boundaries

Scale space

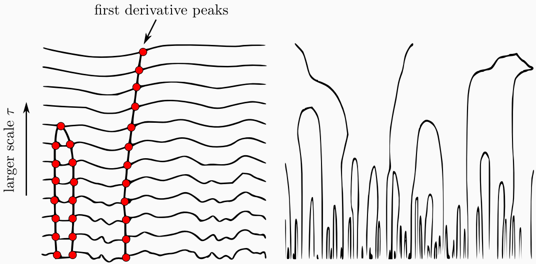
Linear scale space

- Solutions of the heat equation define a linear scale space,
- Each scale is a linear transform/convolution of the previous one.
- Recall that Gaussians have a multi-scale property: $\mathcal{G}_{\gamma^2} * \mathcal{G}_{\gamma^2} = \mathcal{G}_{2\gamma^2}$.

Scale space

Linear scale space

- Solutions of the heat equation define a linear scale space,
- Each scale is a linear transform/convolution of the previous one.
- Recall that Gaussians have a multi-scale property: $\mathcal{G}_{\gamma^2} * \mathcal{G}_{\gamma^2} = \mathcal{G}_{2\gamma^2}$.



- Define an edge as a local extremum of the first derivative [Witkin, 1983]
  1. Edge location is not preserved across the scale space,
  2. Two edges may merge with increasing size,
  3. An edge may not split into two with increasing size.

Scale space

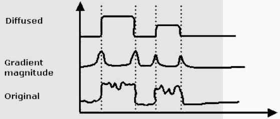
- Nonlinear filters (e.g., median filters) can be used to generate a scale-space,
- But, they usually violate the causality condition.

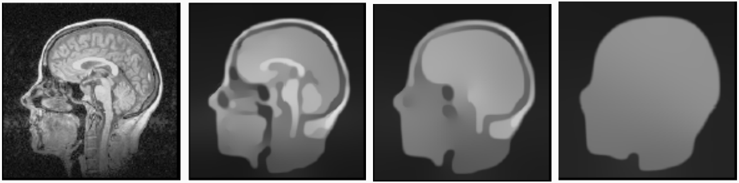
Scale space

- Nonlinear filters (e.g., median filters) can be used to generate a scale-space,
- But, they usually violate the causality condition.

Non-linear scale space

- Immediate localization: fixed edge locations
- Piece-wise smoothing: diffuse between boundaries





At all scales the image will consist of smooth regions separated by edges.
How to build such a scale-space?

Anisotropic diffusion

Towards non-linear diffusion

The conductivity α controls the amount of smoothing per time unit

$$\frac{\partial x}{\partial t} = \alpha \Delta x \quad \equiv \quad x(s,t) = y * \mathcal{G}_{2\alpha t}$$

Image-dependent conductivity

$$\Delta = \frac{\partial^2}{\partial s_1^2} + \frac{\partial^2}{\partial s_2^2} = \begin{pmatrix} \frac{\partial}{\partial s_1} & \frac{\partial}{\partial s_2} \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial s_1} \\ \frac{\partial}{\partial s_2} \end{pmatrix} = \nabla^T \nabla = \operatorname{div} \nabla$$

- Rewrite the heat equation as

$$\frac{\partial x}{\partial t} = \operatorname{div}(\alpha \nabla x)$$

- Basic ideas:
  - make α evolve with space/time in order to preserve edges,
  - set $\alpha = 0$ around edges, and $\alpha > 0$ inside regions,
  - encourage intra-region smoothing,
  - and discourage inter-region smoothing.

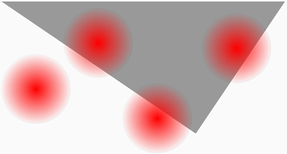
Anisotropic diffusion [Perona and Malik, 1990]

$$\frac{\partial x}{\partial t} = \operatorname{div}(\underbrace{g(\|\nabla x\|_2^2)}_{\alpha} \nabla x) \quad \text{with} \quad x(s_1, s_2, 0) = y(s_1, s_2)$$

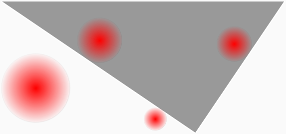
where $g : \mathbb{R}^+ \to [0, 1]$ is decreasing and satisfies

$$g(0) = 1 \quad \text{and} \quad \lim_{u \to \infty} g(u) = 0.$$

- Inside regions with small gradient: fast isotropic diffusion,
- Around edges with large gradients: small diffusion,
- In fact isotropic, sometimes referred to as inhomogeneous diffusion.



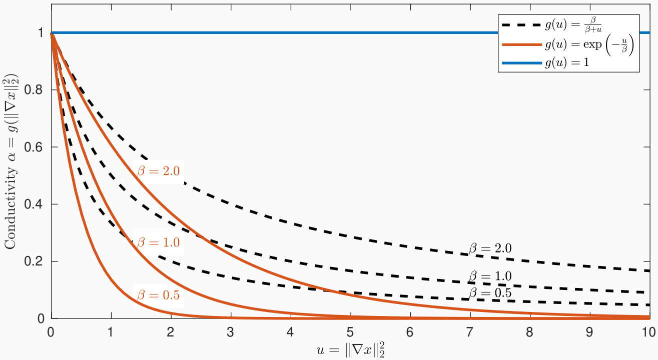
(a) Heat equation / linear diffusion



(b) Inhomogeneous diffusion

Common choices (for $\beta > 0$):

$$g(u) = \frac{\beta}{\beta + u} \quad \text{or} \quad g(u) = \exp\left(-\frac{u}{\beta}\right)$$



Anisotropic diffusion – Variants

Regularized Perona-Malik model [Catté, Lions, Morel, Coll, 1992]

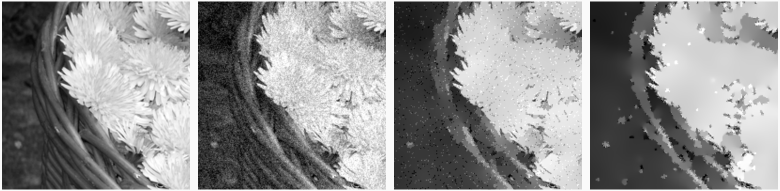
- Classical Perona-Malik solution may be ill-posed:

  The PDE may have no solution or an infinite number of solutions,
  $\Rightarrow$ In practice: small perturbations in y lead to strong deviations.

- Idea: smooth the conductivity field at a small cost of localization

$$\frac{\partial x}{\partial t} = \mathrm{div}(g(\|\nabla(\mathcal{G}_\sigma * x)\|_2^2)\nabla x)$$

where $\mathcal{G}_{\sigma^2}$ is a small Gaussian kernel of width $\sigma > 0$.



(c) x_0 (d) $y = x_0 + w$ (e) x^{400} (AD) (f) x^{400} (R-AD)

Anisotropic diffusion – Resolution schemes

General diffusion model

$$\frac{\partial x}{\partial t} = A(x)x$$

with
$$\begin{cases} \bullet \text{ Heat equation:} & A(x) = \Delta = \operatorname{div} \nabla \\ \bullet \text{ Perona-Malik:} & A(x) = \operatorname{div} g(\|\nabla x\|_2^2)\nabla \\ \bullet \text{ Reg. Perona-Malik:} & A(x) = \operatorname{div} g(\|\nabla (\mathcal{G}_\sigma * x)\|_2^2)\nabla \end{cases}$$

Except for the heat equation,
no explicit continuous solutions in general.

Anisotropic diffusion – Resolution schemes

Resolution schemes: discretization in time

1. Explicit: $x^{k+1} = (\mathrm{Id} + \gamma A(x^k))x^k$ (direct)
2. Semi-implicit: $x^{k+1} = (\mathrm{Id} - \gamma A(x^k))^{-1}x^k$ (linear system to invert)
3. Fully-implicit: $x^{k+1} = (\mathrm{Id} - \gamma A(x^{k+1}))^{-1}x^k$ (nonlinear)

Because A depends on x^k, these are not geometric progressions.

- Need to be run iteratively,
- For explicit scheme: $\left\{ \begin{array}{l} \bullet \text{ Same CFL conditions } \gamma < \frac{1}{2d} \\ \Rightarrow \text{ at least } O(n^2) \text{ for } k \text{ to reach time } m. \end{array} \right.$

Example (Explicit scheme for R-AD)

$$x^{k+1} = x^k + \gamma \operatorname{div}(g(\|\nabla(\mathcal{G}_\sigma * x^k)\|_2^2)\nabla x^k)$$

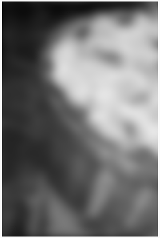
with $g : \mathbb{R} \to \mathbb{R}$ and $\gamma < \frac{1}{2d}$

```python
g  = lambda u: beta / (beta + u)
nu = im.kernel('gaussian', tau=sigma, s1=2, s2=2)

# Explicit scheme for regularized anisotropic diffusion
x = y
for k in range(m):
    x_conv = im.convolve(x, nu)
    alpha  = g(im.norm2(im.grad(x_conv)))
    x      = x + gamma * im.div(alpha * im.grad(x))
```

Anisotropic diffusion – Explicit scheme – Results





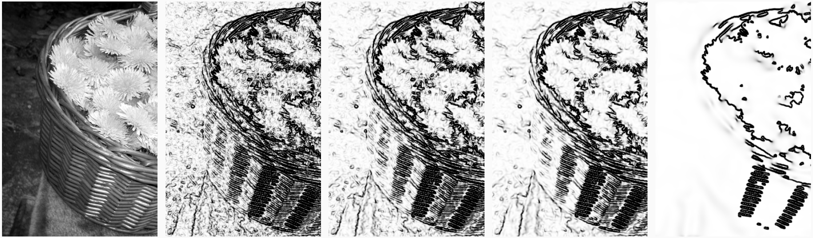


(a) x_0 (b) x^5 (heat) (c) x^{15} (heat) (d) x^{30} (heat) (e) x^{300} (heat)





(f) $y = x_0 + w$ (g) x^5 (R-AD) (h) x^{15} (R-AD) (i) x^{30} (R-AD) (j) x^{300} (R-AD)



(a) x_0 (b) g^5 (R-AD) (c) g^{15} (R-AD) (d) g^{30} (R-AD) (e) g^{300} (R-AD)

(f) $y = x_0 + w$ (g) x^5 (R-AD) (h) x^{15} (R-AD) (i) x^{30} (R-AD) (j) x^{300} (R-AD)

Example (Implicit scheme)

$$x^{k+1} = (\mathrm{Id} - \gamma A(x^k))^{-1} x^k \quad \text{and} \quad \text{converges for any } \gamma > 0$$

Naive idea

- At each iteration k, build the matrix $\boldsymbol{M} = \mathrm{Id} - \gamma A(x^k)$
- Invert it with the function `inv` of Python.

Problem of the naive idea (1/2)

- $\boldsymbol{M}$ is a $n \times n$ matrix,
- If your image is $n = 1024 \times 1024$ (8Mb), this will require

$$\texttt{sizeof(double)} \times n \times n = 8 \cdot 2^{40} = 8\text{Tb}$$

Problem of the naive idea (2/2)

- Best case scenario, you have a few Gb of RAM:

  Python stops and says "`Out of memory`"

- Not too bad scenario, you have more than 8Tb of RAM:

  computation takes forever (in general $O(n^3)$) $\longrightarrow$ kill Python

- Worst case scenario, you have less but close to 8Tb of RAM:

  OS starts swapping and is non-responsive $\longrightarrow$ hard reboot

Anisotropic diffusion – Semi-implicit scheme

Take home message

- When we write on paper $y = \boldsymbol{M}x$ (with x and y images), in your code:

  never

Take home message

- When we write on paper $y = \boldsymbol{M}x$ (with x and y images), in your code:

  never, never

Anisotropic diffusion – Semi-implicit scheme

Take home message

- When we write on paper $y = \boldsymbol{M}x$ (with x and y images), in your code:

  never, never, never

Take home message

- When we write on paper $y = \boldsymbol{M}x$ (with x and y images), in your code:

  never, never, never, **never build the matrix $\boldsymbol{M}$**

Take home message

- When we write on paper $y = \boldsymbol{M}x$ (with x and y images), in your code:

  never, never, never, **never build the matrix $\boldsymbol{M}$**

- What is the alternative?
  - Use knowledge on the structure of $\boldsymbol{M}$ to compute $y = \boldsymbol{M}x$ quickly
  - As for the FFT: $\boldsymbol{F}x = \texttt{fft2}(x)$ (you never had to build $\boldsymbol{F}$)
  - If $\boldsymbol{M} = \frac{1}{n}\begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & & & \\ 1 & 1 & \dots & 1 \end{pmatrix}$, how do I compute $\boldsymbol{M}x$ in $O(n)$?
  - If $\boldsymbol{M}$ is sparse (# of non-zero entries in $O(n)$), use *sparse* matrices.

Design the operator $z \mapsto \boldsymbol{M}z$ rather than $\boldsymbol{M}$

But how do I compute $x = M^{-1}y$ if I do not build M?

- Solve the system

$$\boldsymbol{M}x = y$$

with a solver that only needs to know the operator $z \mapsto \boldsymbol{M}z$.

Conjugate gradient

- If M is square symmetric definite positive, conjugate gradient solves the system by iteratively evaluating $z \mapsto \boldsymbol{M}z$ at different locations z.
- Use `im.cg`. Example to solve $2x = y$:

```
x = im.cg(lambda z: 2 * z, y)
```

Anisotropic diffusion – Semi-implicit scheme

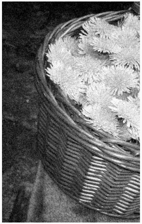
Explicit: $x^{k+1} = (\mathrm{Id} + \gamma A(x^k))x^k$ Implicit: $x^{k+1} = (\mathrm{Id} - \gamma A(x^k))^{-1}x^k$

```
# Explicit vs Implict scheme for regularized anisotropic diffusion
x_e = y
x_i = y
for k in range(m):
    # Explicit (0 < gamma < 0.25)
    x_e = rad_step(x_e, x_e, sigma, gamma, g)

    # Implicit (0 < gamma)
    x_i = im.cg(lambda z: rad_step(x_i, z, sigma, -gamma, g), x_i)
```

```
# One step r = (Id + gamma A(x)) z for the regularized AD
nu = im.kernel('gaussian', tau=sigma, s1=2, s2=2)
def rad_step(x, z, sigma, gamma, g):
    x_conv = im.convolve(x, nu)
    alpha  = g(im.norm2(im.grad(x_conv)))
    r      = z + gamma * im.div(alpha * im.grad(z))
```

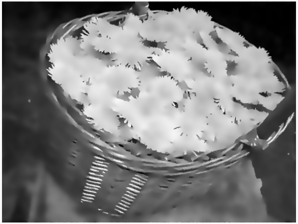
Anisotropic diffusion – Semi-implicit scheme – Results

Input

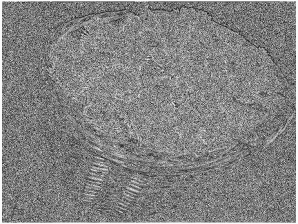


(a) $\sigma = 20$

Explicit

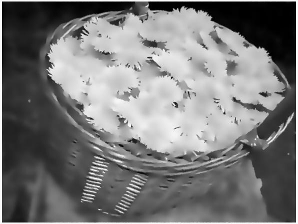


(b) $k = 100, \gamma = 0.24$



(c) $k = 1, \gamma = 0.24 \times 100$

Implicit

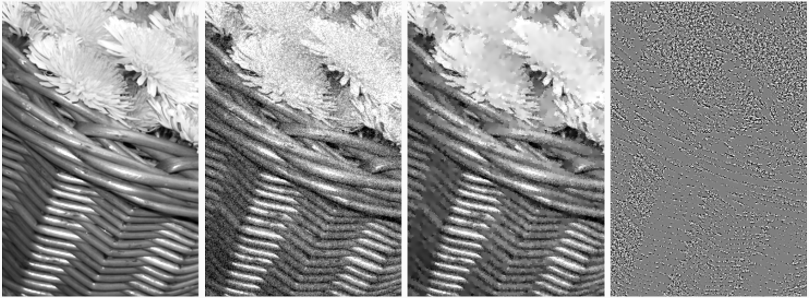


(d) $k = 100, \gamma = 0.24$ ($3\times$ slower)

(e) $k=1, \gamma=0.24\times 100$ ($2\times$ faster)

(Note: $\boldsymbol{M}$ also block tri-diagonal $\Rightarrow$ Thomas algorithm can be used and is even faster)

Anisotropic diffusion – Limitations



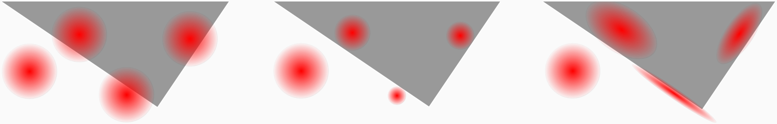
(a) x_0 (original) (b) $y = x_0 + w$ (c) x (Perona-Malik) (d) $y - x$ (method noise)

Behavior

- Inside regions with small gradient magnitude: fast isotropic smoothing.
- Diffusion stops around strong image gradients (structure-preserving).
- Noise on edges is not reduced by Perona-Malik solutions.

Can we be really anisotropic?

Anisotropic diffusion – Truly anisotropic behavior?



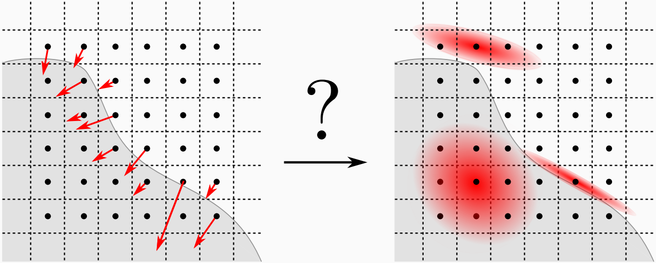
(a) Homogeneous (b) Inhomogeneous (c) Anisotropic

- Make neighborhoods truly anisotropic.
- Reminder: ellipses in 2d = encoded by a 2×2 sdp matrix (rotation + re-scaling)
- Replace the conductivity by a matrix-valued function

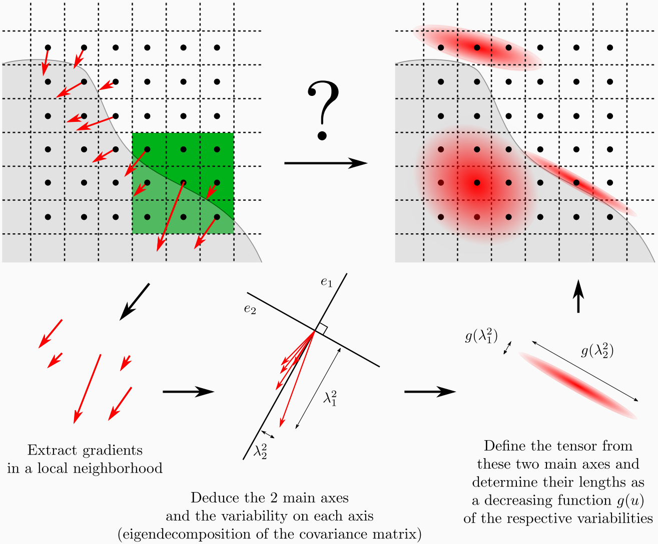
$$\frac{\partial x}{\partial t} = \operatorname{div}(\underbrace{T(x)\nabla x}_{\text{matrix vector product}}).$$

- T maps each pixel position of x to a 2×2 matrix.
- $T(x)$ is called a tensor field,
- The function T should control the direction of the flow.

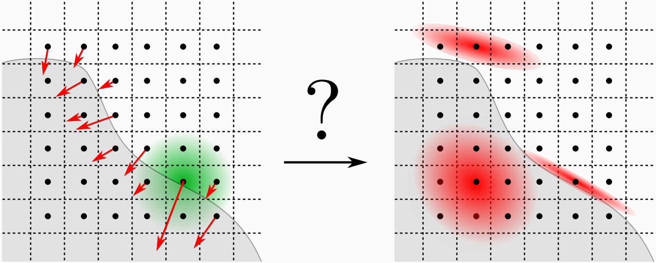
Anisotropic diffusion – Truly anisotropic behavior [Weickert, 1999]



Anisotropic diffusion – Truly anisotropic behavior [Weickert, 1999]



Anisotropic diffusion – Truly anisotropic behavior [Weickert, 1999]



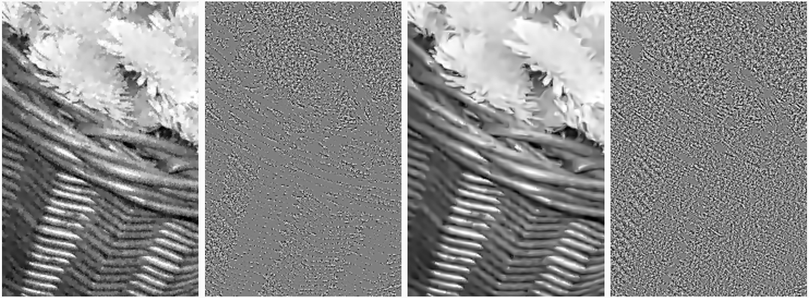
$$\frac{\partial x}{\partial t} = \operatorname{div}(\underbrace{T(x)\nabla x})$$

where $\quad T(x) = h[\underbrace{\mathcal{G}_\rho * ((\nabla \mathcal{G}_\sigma * x)(\nabla \mathcal{G}_\sigma * x)^T)}_{\text{local covariance matrix}}]$

with $\quad \underbrace{h\left[\boldsymbol{E}\begin{pmatrix}\lambda_1^2 & \\ & \lambda_2^2\end{pmatrix}\boldsymbol{E}^{-1}\right] = \boldsymbol{E}\begin{pmatrix}g(\lambda_1^2) & \\ & g(\lambda_2^2)\end{pmatrix}\boldsymbol{E}^{-1}}_{\text{decreasing (matrix-valued) function of the eigenvalues}}$

and $\quad \boldsymbol{E} = \begin{pmatrix}e_1 & e_2\end{pmatrix} \quad \longleftarrow \quad$ eigenvectors

Anisotropic diffusion – Comparison



(a) x (P-M., 1990) (b) $y - x$ (method noise) (c) x (Weickert, 1999) (d) $y - x$ (method noise)

Behavior

- Inside regions with small gradient magnitude: fast smoothing,
- Around objects: diffusion aligns to anisotropic structures,
- Noise on edges reduced compared to inhomogeneous isotropic diffusion.

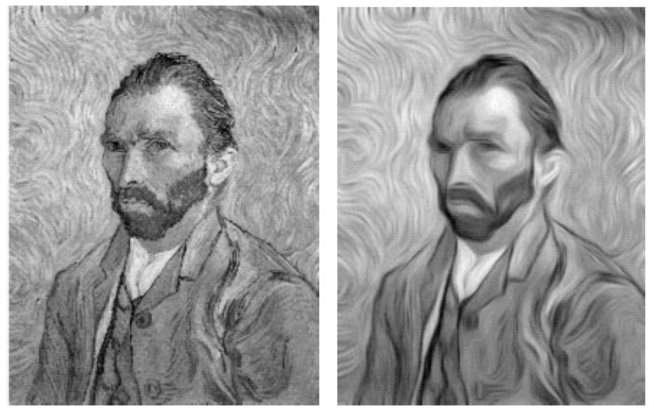


Figure 1 – (left) input y. (right) truly anisotropic diffusion

Source: A. Roussos

Anisotropic diffusion – Illustrations

Figure 2 – (left) input y. (middle) inhomogeneous diffusion. (right) truly anisotropic.

Source: A. Roussos

Anisotropic diffusion – Illustrations

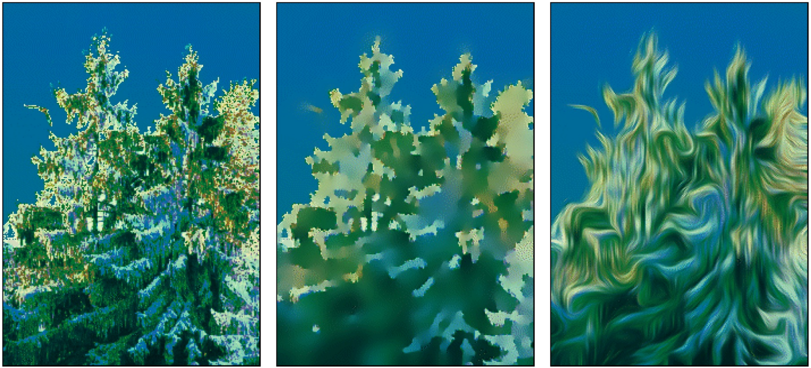


Figure 3 – (left) input y. (middle) inhomogeneous diffusion. (right) truly anisotropic.

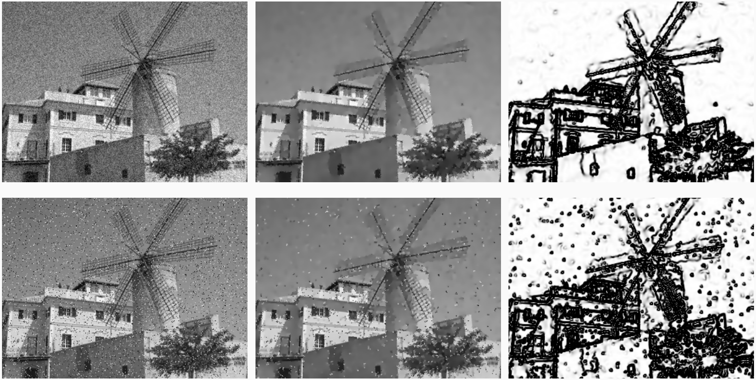
Source: A. Roussos

Anisotropic diffusion – Remaining issues

- When to stop the diffusion?
- How to use that for deblurring / super-resolution / linear inverse problems?
- Non-adapted for non-Gaussian noises (e.g., impulse noise).

Gaussian noise

Impulse noise



(a) Input image (b) Perona-Malik (c) Conductivity

Variational methods

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

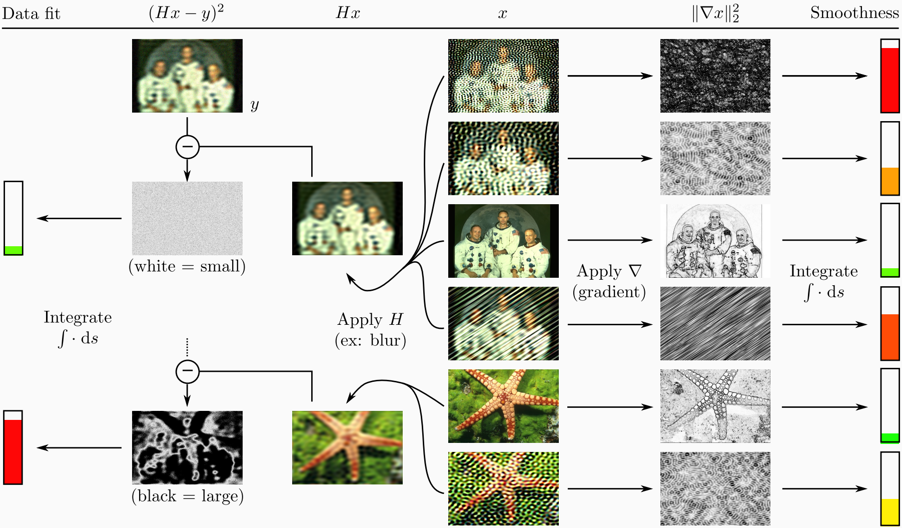
$$F(x) = \frac{1}{2} \int_\Omega (H(x)(s) - y(s))^2 + \tau \|\nabla x(s)\|_2^2 \, \mathrm{d}s$$

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

Variational methods - Tikhonov functional

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

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) $\tau \to \infty$

Variational methods

How to solve this variational problem?

2 solutions:

1. Functional $\longrightarrow$ Discretization $\longrightarrow$ Numerical scheme

2. Functional $\longrightarrow$ PDE $\longrightarrow$ Discretization & Euler schemes

Variational methods

How to solve this variational problem?

2 solutions:

1. Functional $\longrightarrow$ Discretization $\longrightarrow$ Numerical scheme

2. Functional $\longrightarrow$ PDE $\longrightarrow$ Discretization & Euler schemes
   (we won't discuss it, *cf.*, Euler-Lagrange equation)

Variational methods – Smooth optimization

Discretization of the functional

$$\min_x \left\{ F(x) = \sum_{k=1}^{n} f(k, x, \nabla x) \right\}$$

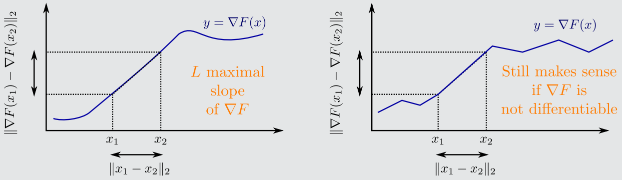
- n: number of pixels,
- k: pixel index, corresponding to location s_k,
- $x \in \mathbb{R}^n$: discrete image,
- ∇x: discrete image gradient,
- $F : \mathbb{R}^n \to \mathbb{R}$: function of a vector.

- Classical optimization problem,
- Look for a vector x that cancels the gradient of F,
- If no explicit solutions, use gradient descent.

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.

Variational methods – Smooth optimization

Gradient descent

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

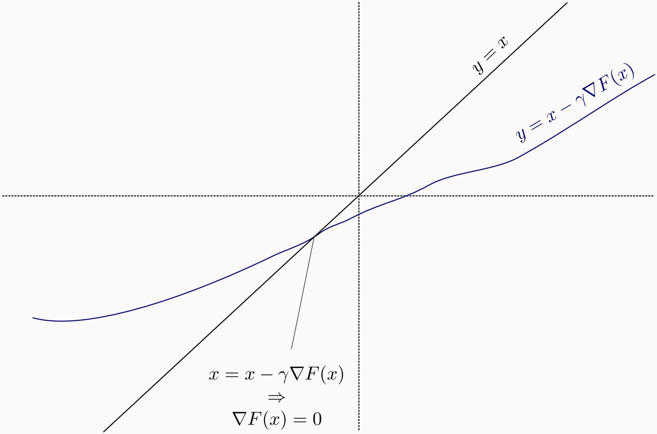
$$x^{k+1} = x^k - \gamma \nabla F(x^k) \ ,$$

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

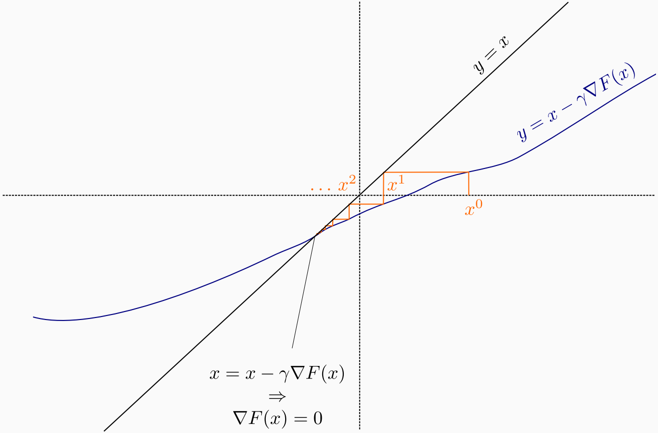
$$\nabla F(x^\star) = 0 \ .$$

- 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 $O(1/k)$.

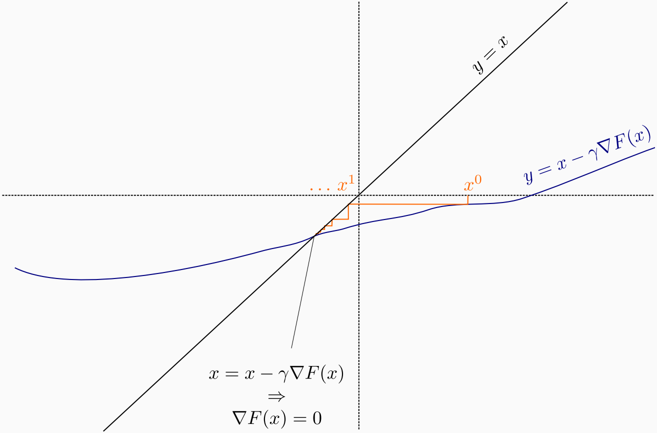
Variational methods – Smooth optimization



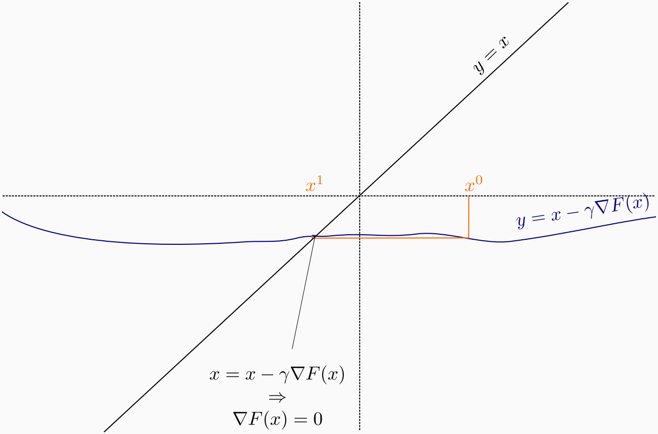
These two curves cross at $x^\star$ such that $\nabla F(x^\star) = 0$



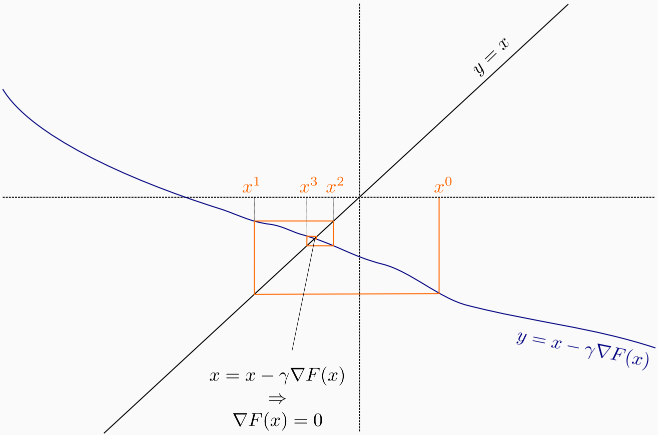
Here γ is small: slow convergence



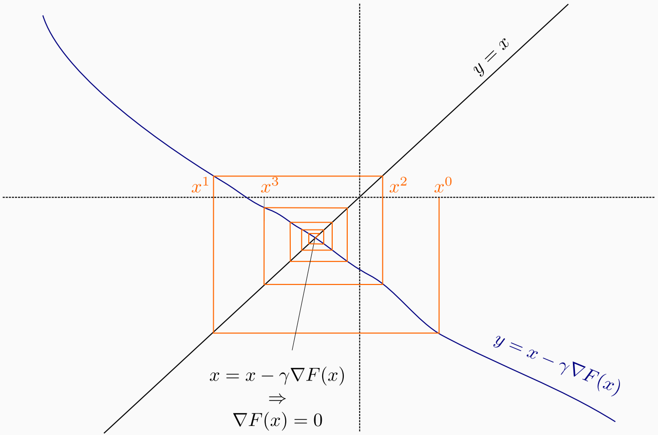
γ a bit larger: faster convergence



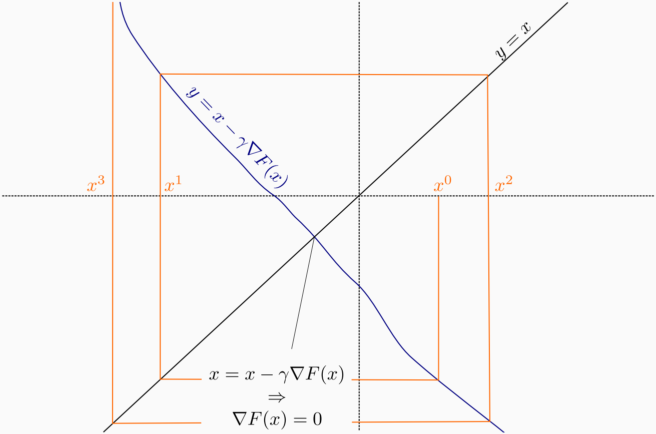
$\gamma \approx 1/L$ even larger: around fastest convergence



γ a bit too large: convergence slows down



γ too large: convergence too slow again



$\gamma > 2/L$: divergence

Gradient descent for convex function

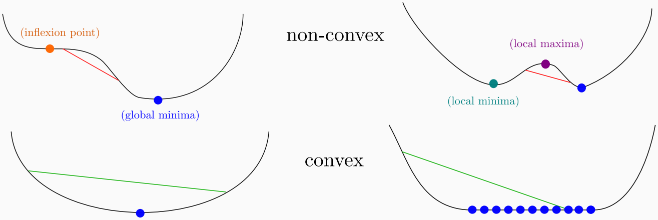
- If moreover F is convex

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

  then, the gradient descent converges towards a global minimum

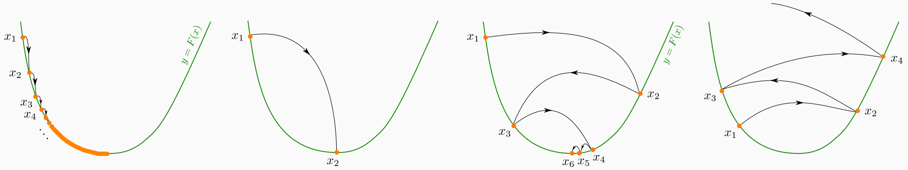
$$x^\star \in \underset{x}{\operatorname{argmin}}\; F(x).$$

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



Variational methods – Smooth optimization

One-dimension



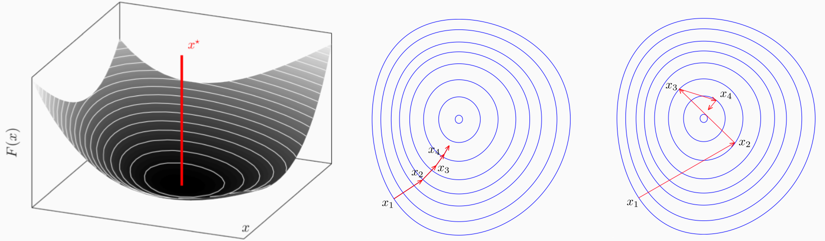
Small step size
Slow convergence

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

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

- $\ell_{2,2}$/Frobenius norm of a matrix:

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

- 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 $\|\Delta\|_2 = 4d$.

- For $\tau \to \infty$ and $x^0 = y$, this converges to the explicit Euler scheme for the Heat equation. The condition $\gamma < \frac{2}{L}$ is equivalent to the CFL condition.

Solutions of the Heat equation tend to minimize the smoothing term.

This explains why at convergence the Heat equation provides constant solutions (when using periodical boundary solutions).

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))

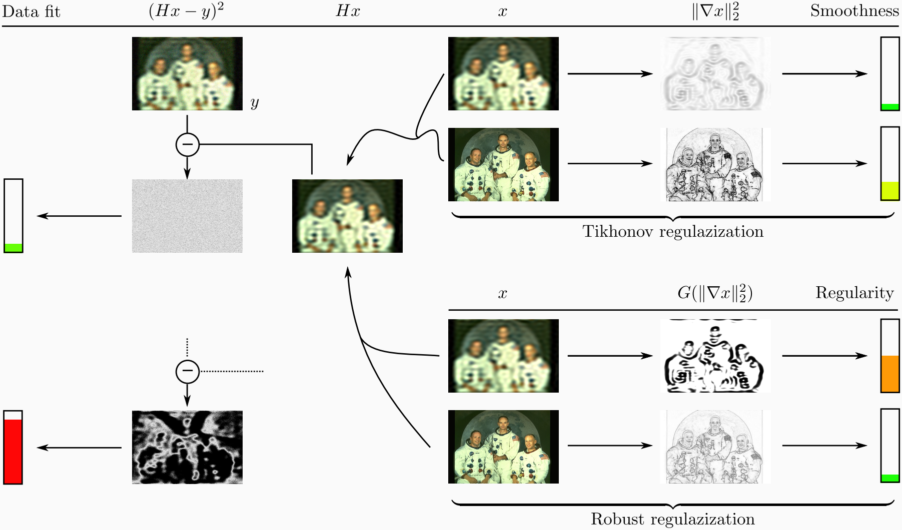
- Explicit solution

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

- 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).

How to incorporate inhomogeneity à la Perona-Malik?

Variational methods - Robust regularization



Use robust regularizers to pick the good candidate

Example (Robust regularization (1/2))

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

- After discretization, its gradient is given by

$$\nabla F(x) = \boldsymbol{H}^*(\boldsymbol{H}x - y) - \tau \operatorname{div}(g(\|\nabla x\|_2^2)\nabla x)$$

  where $g(u) = G'(u)$.

- The gradient descent becomes

$$x^{k+1} = x^k - \gamma(\underbrace{\boldsymbol{H}^*(\boldsymbol{H}x^k - y)}_{\text{retroaction}} - \tau \operatorname{div}(g(\|\nabla x^k\|_2^2)\nabla x^k)\;.$$

Without the retroaction term ($\tau \to \infty$),
this is exactly the explicit Euler scheme for the anisotropic diffusion.

(a) Low resolution y

Robust regularization for $\times$ 16 super-resolution

(b) Tiny τ (c) Small τ (d) Good τ (e) High τ (f) Huge τ

Variational methods – Robust regularization

(a) Low resolution y

Tikhonov regularization for $\times$ 16 super-resolution

(b) Tiny τ (c) Small τ (d) Good τ (e) High τ (f) Huge τ

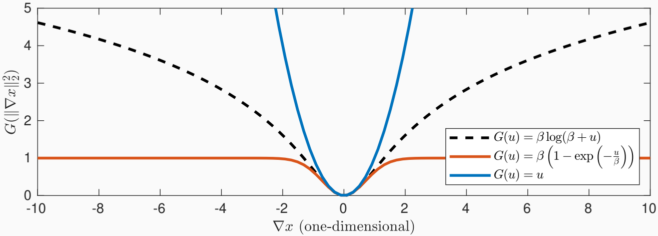
What are the choices of G,
leading to the choice of Perona and Malik?

Example (Robust regularization (2/2))

$$G(u) = u \quad \Rightarrow \quad g(u) = 1 \qquad \text{(Heat)}$$

$$G(u) = \beta \log(\beta + u) \quad \Rightarrow \quad g(u) = \frac{\beta}{\beta + u} \qquad \text{(AD)}$$

$$G(u) = \beta \left(1 - \exp\left(-\frac{u}{\beta}\right)\right) \quad \Rightarrow \quad g(u) = \exp\left(-\frac{u}{\beta}\right) \qquad \text{(AD)}$$



- Tikhonov (blue) is convex:

  $\Rightarrow$ global minimum ☺

  - huge penalization for large gradients: does not allow for edges,

    $\Rightarrow$ smooth solutions. ☹

- The other two are non-convex:

  $\Rightarrow$ stationary point depending on the initialization ☹

  - small penalization for large gradients: allows for edges (robust),

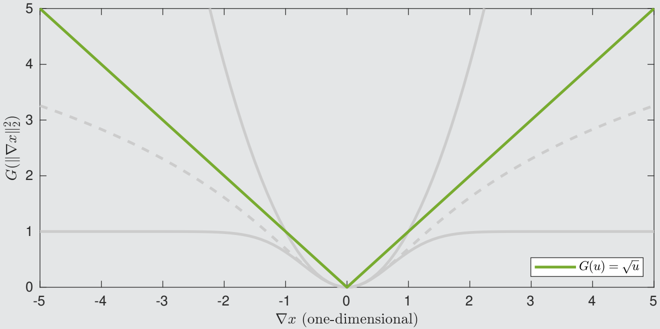
    $\Rightarrow$ sharp solutions. ☺

Total-Variation

Can we take the best of both worlds?

Total-Variation (TV) or ROF model **[Rudin, Osher, Fatemi, 1992]**

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



- Tightest convex penalty.
- Convex, robust and **induces sparsity**.

One-dimensional case

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

1d Total-Variation

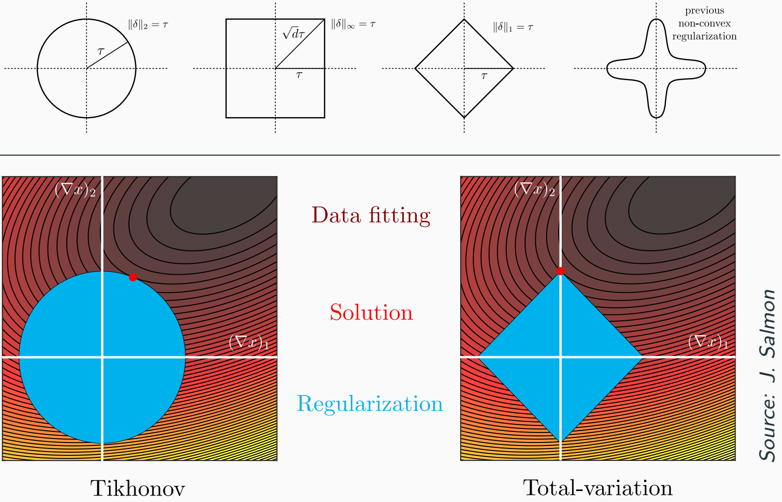
- Its discretization leads to

$$\begin{aligned} F(x) &= \frac{1}{2}\|\boldsymbol{H}x - y\|_2^2 + \frac{\tau}{2} \sum_k |(\nabla x)_k| \\ &= \frac{1}{2}\|\boldsymbol{H}x - y\|_2^2 + \frac{\tau}{2}\|\nabla x\|_1 \end{aligned}$$

- ℓ_p norm of a vector:

$$\|v\|_p = \left(\sum_k |v_k|^p \right)^{1/p}$$

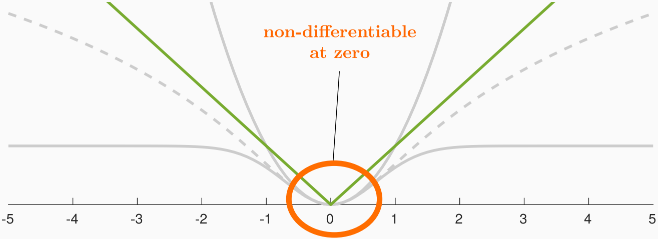
Total-Variation – One-dimensional case



For TV, the gradient will be zero for most of its coordinates.

This is due to the corners of the ℓ_1 ball.

Total-Variation – One-dimensional case

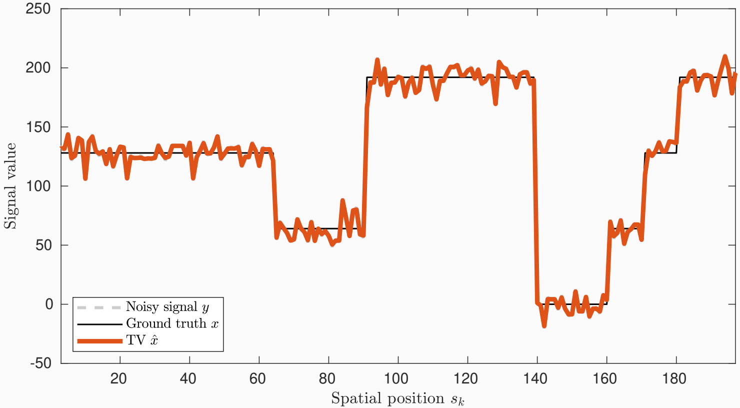


Gradient sparsity

- Sparsity of the gradient $\Leftrightarrow$ **piece wise constant solutions**
- Non-smooth (non-differentiable) $\Rightarrow$ can't use gradient descent. ☹

Large noise reduction with edge preservation
but, **convex non-smooth optimization problem**.

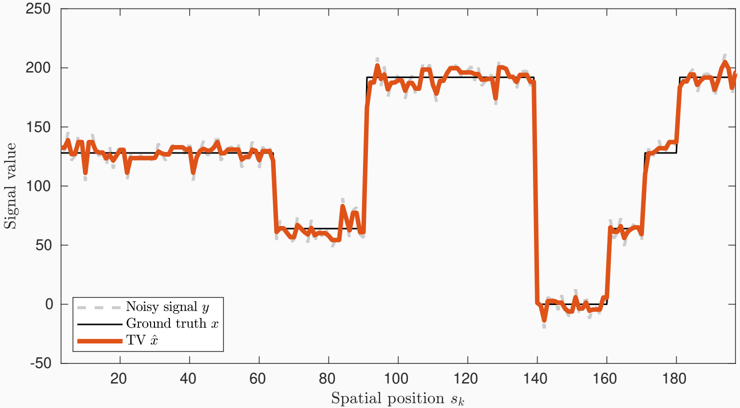
A solution: proximal splitting methods (in a few classes),
kind of implicit Euler schemes.



Evolution with the regularization parameter τ

- Too small: noise overfitting / staircasing,
- Too large: loss of contrast, loss of objects.

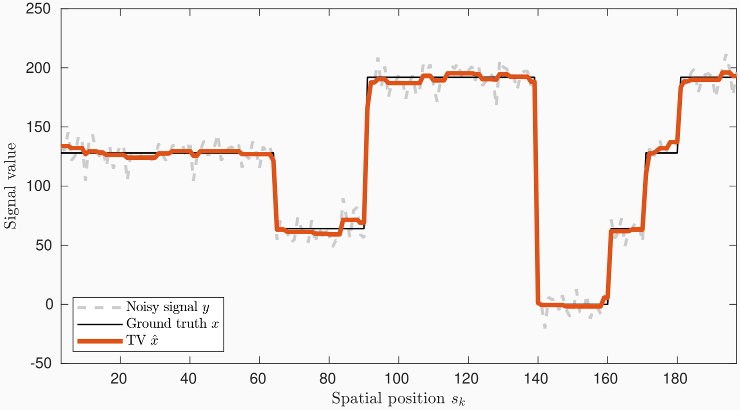
Total-Variation – One-dimensional case (denoising)



Evolution with the regularization parameter τ

- Too small: noise overfitting / staircasing,
- Too large: loss of contrast, loss of objects.

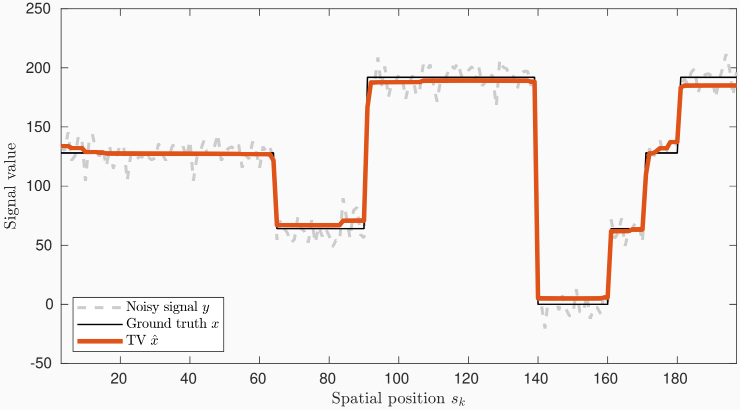
Total-Variation – One-dimensional case (denoising)



Evolution with the regularization parameter τ

- Too small: noise overfitting / staircasing,
- Too large: loss of contrast, loss of objects.

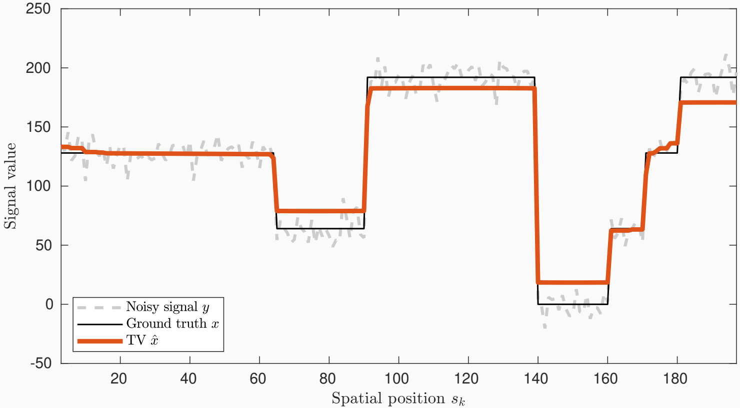
Total-Variation – One-dimensional case (denoising)



Evolution with the regularization parameter τ

- Too small: noise overfitting / staircasing,
- Too large: loss of contrast, loss of objects.

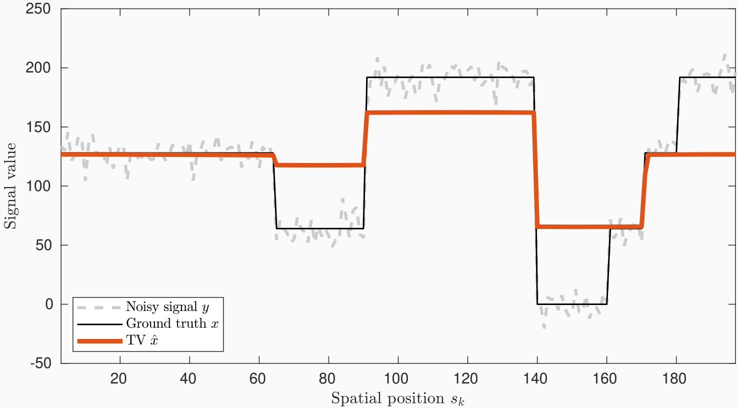
Total-Variation – One-dimensional case (denoising)



Evolution with the regularization parameter τ

- Too small: noise overfitting / staircasing,
- Too large: loss of contrast, loss of objects.

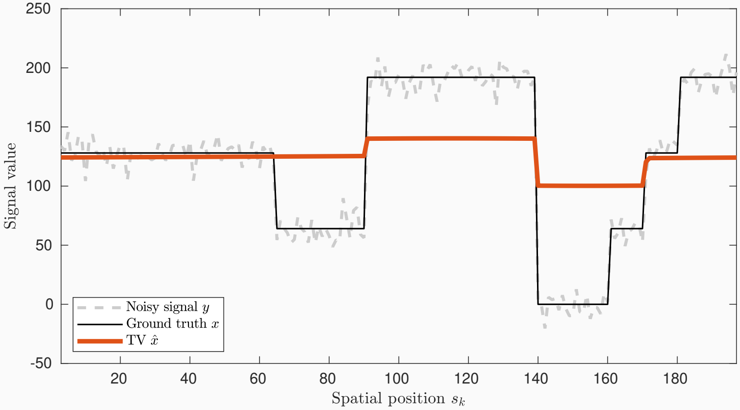
Total-Variation – One-dimensional case (denoising)



Evolution with the regularization parameter τ

- Too small: noise overfitting / staircasing,
- Too large: loss of contrast, loss of objects.

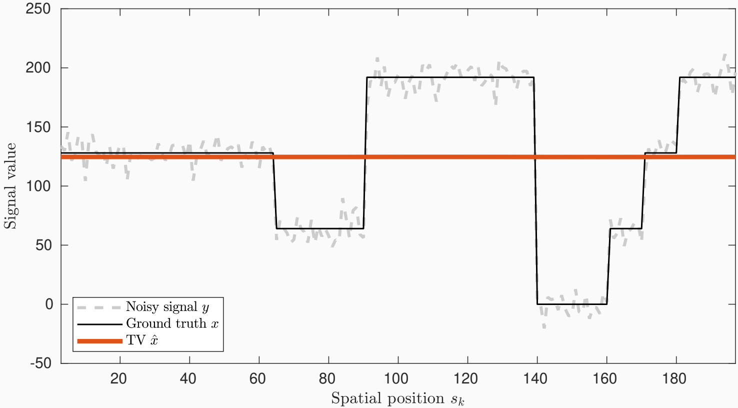
Total-Variation – One-dimensional case (denoising)



Evolution with the regularization parameter τ

- Too small: noise overfitting / staircasing,
- Too large: loss of contrast, loss of objects.

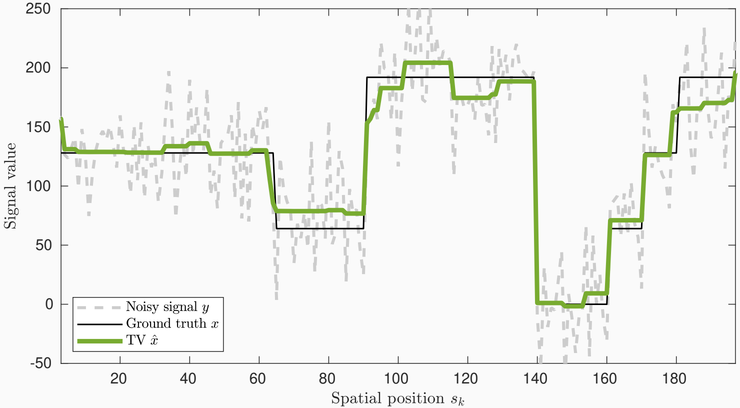
Total-Variation – One-dimensional case (denoising)



Evolution with the regularization parameter τ

- Too small: noise overfitting / staircasing,
- Too large: loss of contrast, loss of objects.

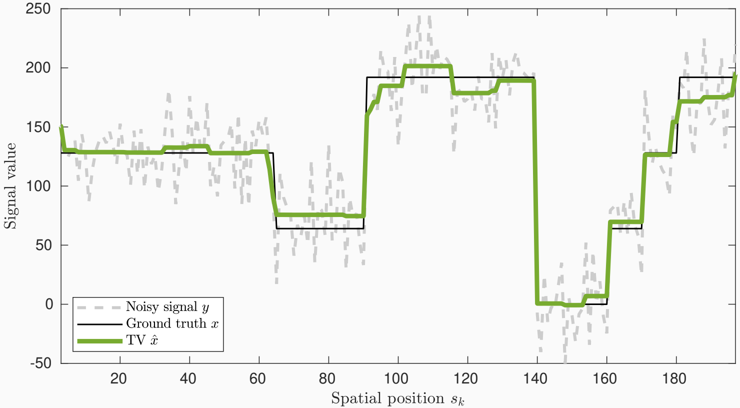
Total-Variation – One-dimensional case (denoising)



Evolution with the noise level σ

- Large noise: staircasing + loss of contrast.
- Small noise: noise reduction + edge preservation

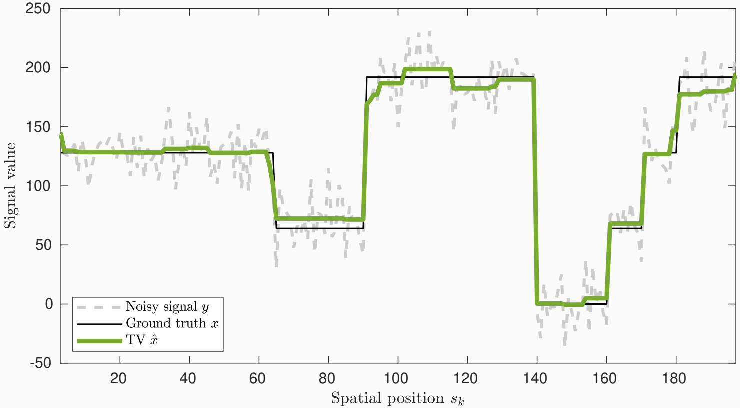
Total-Variation – One-dimensional case (denoising)



Evolution with the noise level σ

- Large noise: staircasing + loss of contrast.
- Small noise: noise reduction + edge preservation

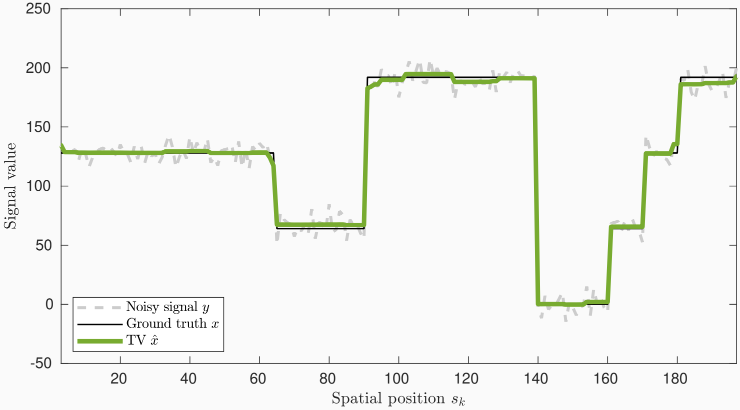
Total-Variation – One-dimensional case (denoising)



Evolution with the noise level σ

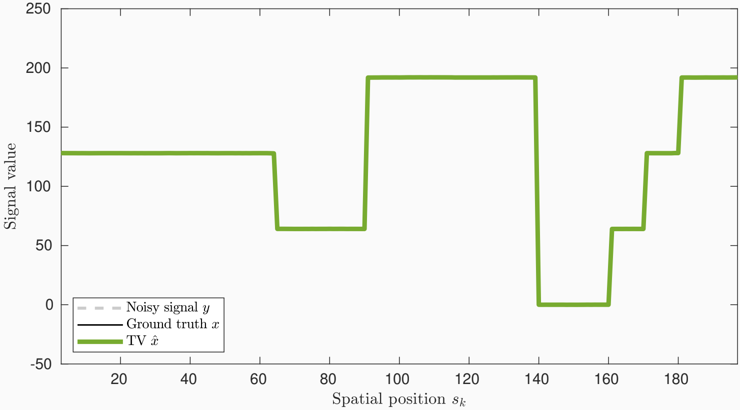
- Large noise: staircasing + loss of contrast.
- Small noise: noise reduction + edge preservation

Total-Variation – One-dimensional case (denoising)



Evolution with the noise level σ

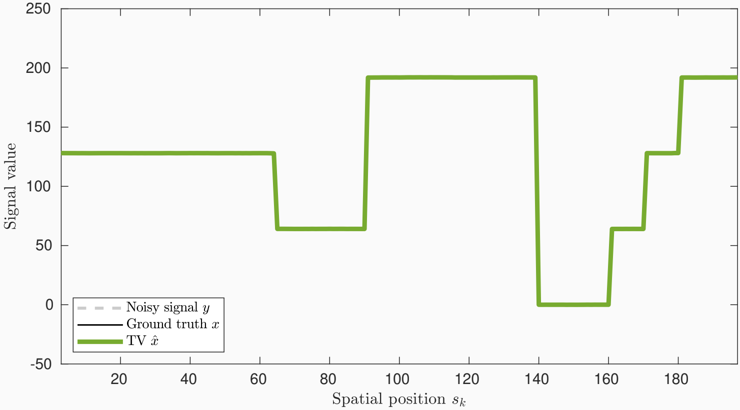
- Large noise: staircasing + loss of contrast.
- Small noise: noise reduction + edge preservation



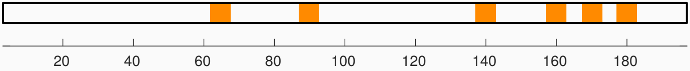
Evolution with the noise level σ

- Large noise: staircasing + loss of contrast.
- Small noise: noise reduction + edge preservation

Total-Variation – One-dimensional case (denoising)



Set of non-zero gradients (jumps) is sparse



Two-dimensional case

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

2d Total-Variation

- Its discretization leads to

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

- $\ell_{p,q}$ norm of a matrix:

$$\|\boldsymbol{A}\|_{p,q} = \left(\sum_k \left(\sum_l |\boldsymbol{A}_{kl}|^p \right)^{q/p} \right)^{1/q}$$

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

(a) Blurry image y

Total-Variation – Results

(b) Tiny τ

Total-Variation – Results

(c) Small τ

Total-Variation – Results

(d) Relatively small τ

(e) Medium τ

(f) Large τ

(g) Even larger τ

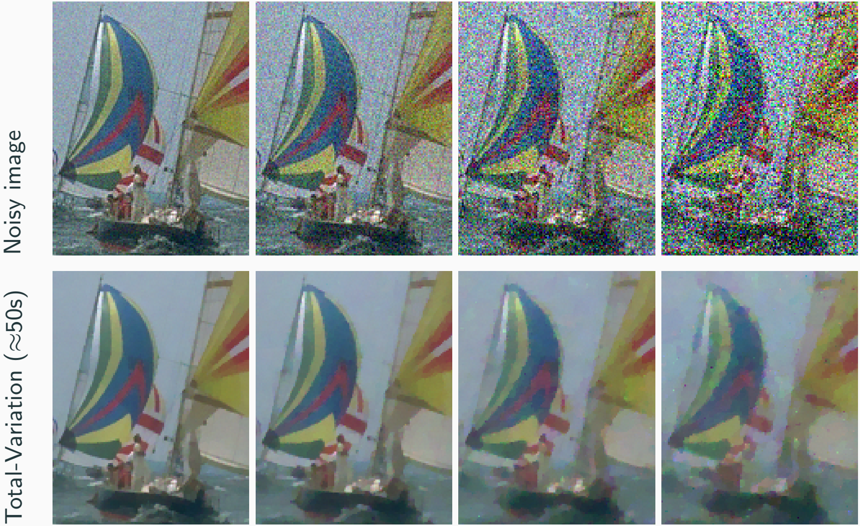
Total-Variation – Results

(h) Too larger τ

Total-Variation – Results

(i) Huge τ

TV regularization for denoising



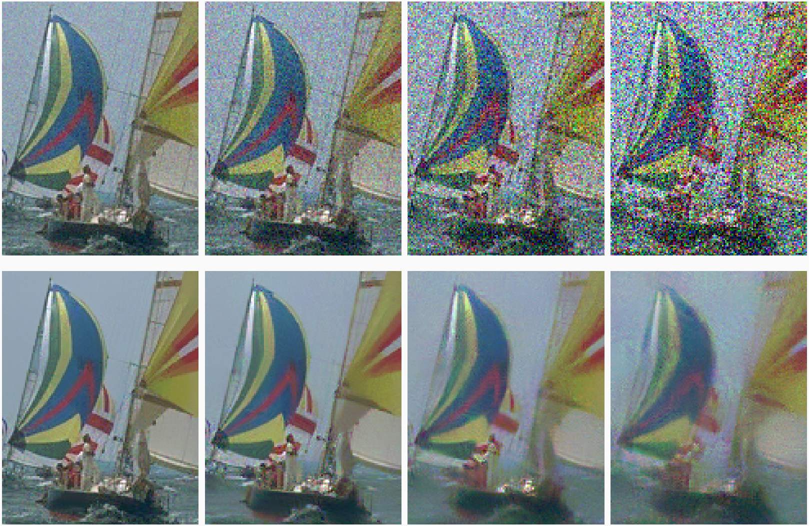
(a) Noise $\sigma = 10$ (b) $\sigma = 20$ (c) $\sigma = 40$ (d) $\sigma = 60$

Total-Variation – Results

TV regularization for denoising

Noisy image

BNL-means (≈30s)



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

Total-Variation – Two-dimensional case

Variant: Anisotropic TV

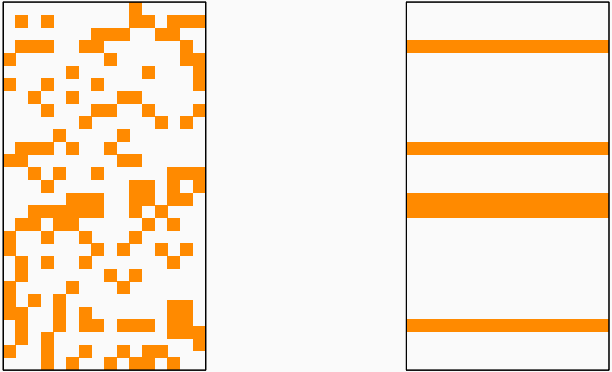
$$F(x) = \frac{1}{2} \int (\boldsymbol{H} x - y)^2 + \tau \|\nabla x\|_1 \, \mathrm{d}s$$

Anisotropic Total-Variation

- Its discretization leads to

$$\begin{aligned} F(x) &= \frac{1}{2}\|\boldsymbol{H} x - y\|_2^2 + \frac{\tau}{2} \sum_k \|(\nabla x)_k\|_1 \\ &= \frac{1}{2}\|\boldsymbol{H} x - y\|_2^2 + \frac{\tau}{2} \|\nabla x\|_{1,1} \end{aligned}$$

- Anisotropic behavior:
    - Penalizes more the gardient in diagonal directions,
    - Favor horizontal and vertical structures,
    - By opposition the $\ell_{2,1}$ version is called Isotropic TV.



(a) Sparsity induced by $\|\boldsymbol{A}x\|_{1,1}$

$\Rightarrow$ **many zero entries**

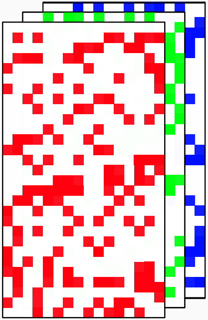
(b) Group sparsity induced by $\|\boldsymbol{A}x\|_{2,1}$

$\Rightarrow$ **many zero rows**

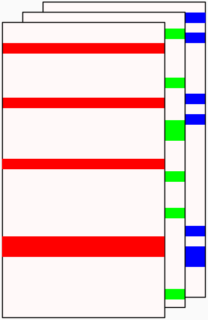
Anisotropic TV: components of the gradient of each pixel are independent.

Isotropic TV: the two components of the gradient are **grouped together**.

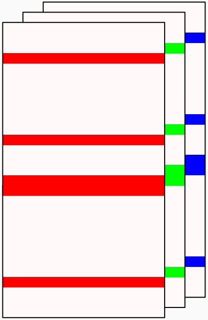
Total-Variation – Two-dimensional case



(a) Independent



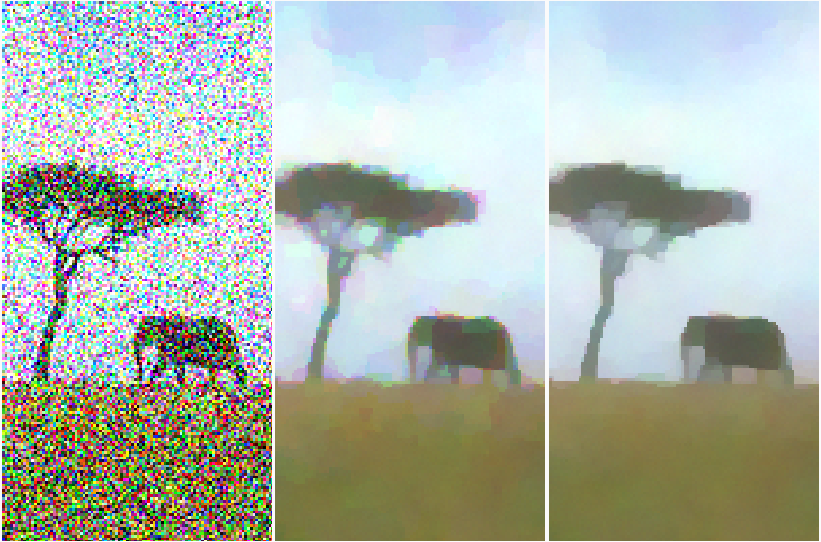
(b) Blocks of gradients



(c) Gradients and colors

We can also group the colors to avoid color aberrations.

Total-Variation – Two-dimensional case



(a) Noisy image (b) Anisotropic TV (c) Anisotropic TV + Color

Total-Variation – Two-dimensional case

(a) Noisy image (b) Isotropic TV (c) Isotropic TV + Color

Total-Variation – Remaining issues

- What to choose for the regularization τ?
- Loss of textures (high frequency objects)
  $\longrightarrow$ images are not piece-wise constant,
- Non-adapted for non-Gaussian noises (e.g., impulse noise).

(a) Gaussian noise (b) TV result (c) Impulse noise (d) TV result

Variational methods – Further reading

For further reading

- Variational methods for image segmentation:
  - Mumford-Shah functional (1989),
  - Active contours / Snakes (Kass et al, 1988),
  - Chan-Vese functional (2001).
- Link with Gibbs priors and Markov Random Fields (MRF):
  - Geman & Geman model (1984),
  - Graph cuts (Boykov, Veksler, Zabih, 2001), (Ishikawa, 2003),

    $\longrightarrow$ Applications in Computer-Vision.
- For more evolved regularization terms:
  - Fields of Experts (Roth & Black, 2008).
  - Total-Generalized Variation (Bredies, Kunisch, Pock, 2010).
- Link with Machine learning / Sparse regression:
  - LASSO (Tibshirani, 1996) / Fused LASSO / Group LASSO.

Questions?

Next class: Bayesian methods

Sources, images courtesy and acknowledgment

L. Condat

A. Horodniceanu

I. Kokkinos

G. Rosman

A. Roussos

J. Salmon

Wikipedia