Energy Minimization and an Introduction to the Bayesian Framework

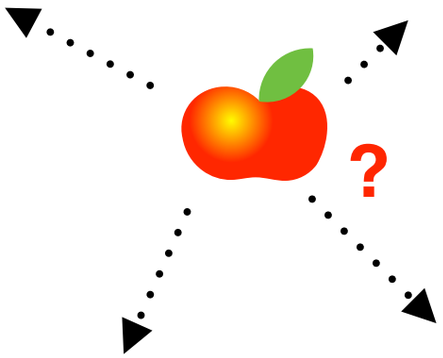
Paolo Favaro

Computer Vision Course

Modeling vs Estimation

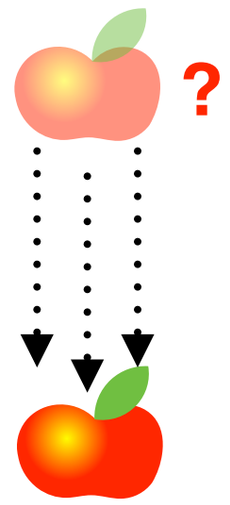
1. Finding models, laws, and/or properties of data

   - E.g., object free-fall $x = gt^2 + v_0 t + x_0$



2. Finding latent parameters of given models and data (**Direct/Inverse problems**)

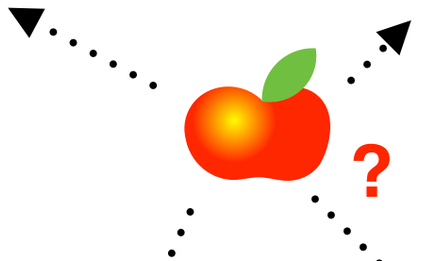
   - E.g., x_0 and v_0 given $x(t)$ and t



Modeling vs Estimation

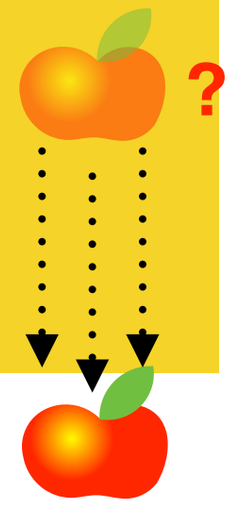
1. Finding models, laws, and/or properties of data

   - E.g., object free-fall $x = gt^2 + v_0 t + x_0$



2. Finding latent parameters of given models and data (**Direct/Inverse problems**)

   - E.g., x_0 and v_0 given $x(t)$ and t



What is an inverse problem?

- **Definition**: *A direct problem is a problem directed towards a loss of information*

- The solution of a direct problem has less information about a physical quantity than the input

Example: Heat propagation

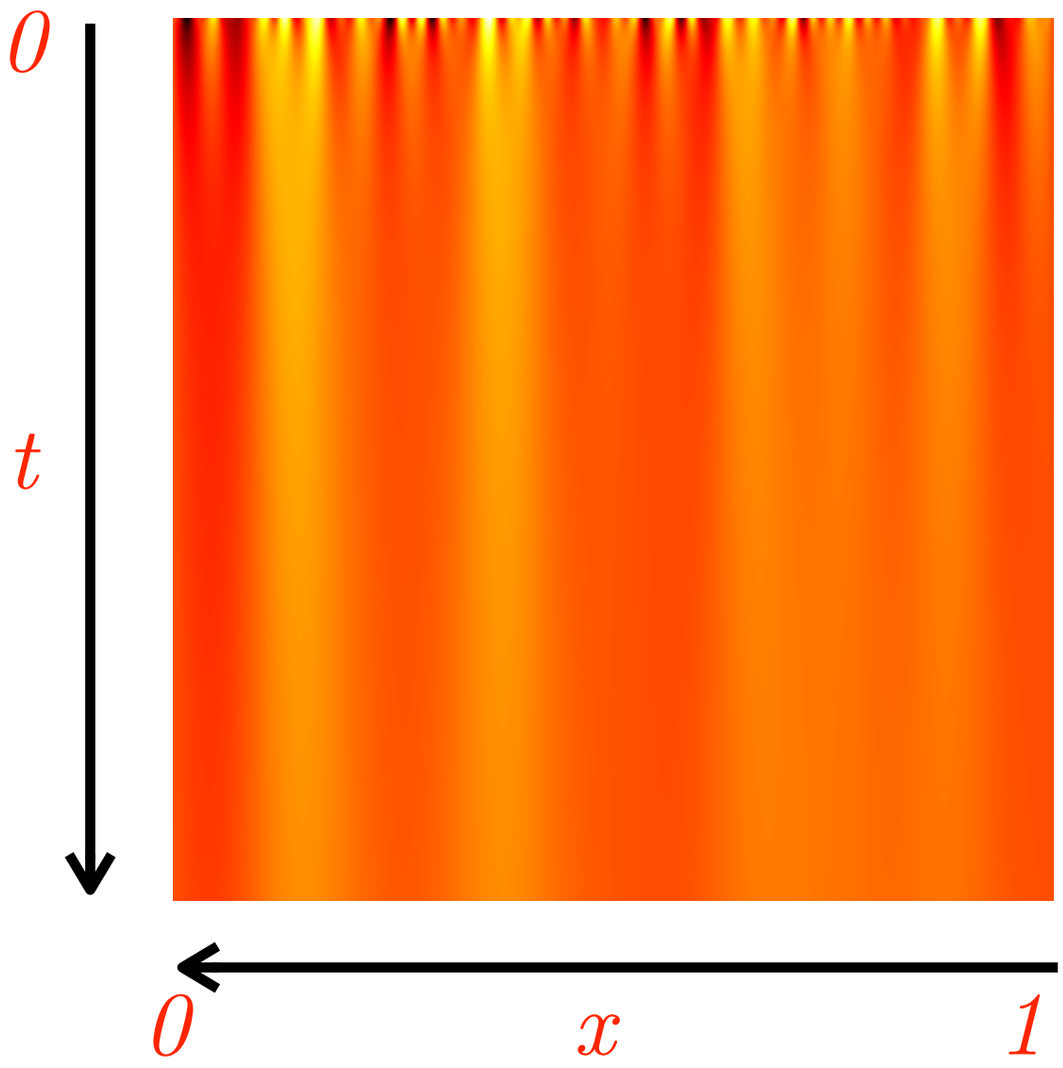
Direct problem: Compute the temperature f at a time $t>0$, given the temperature at $t=0$.

Heat propagation follows

$$\frac{\partial^2 f}{\partial x^2} = \frac{1}{D}\frac{\partial f}{\partial t}$$

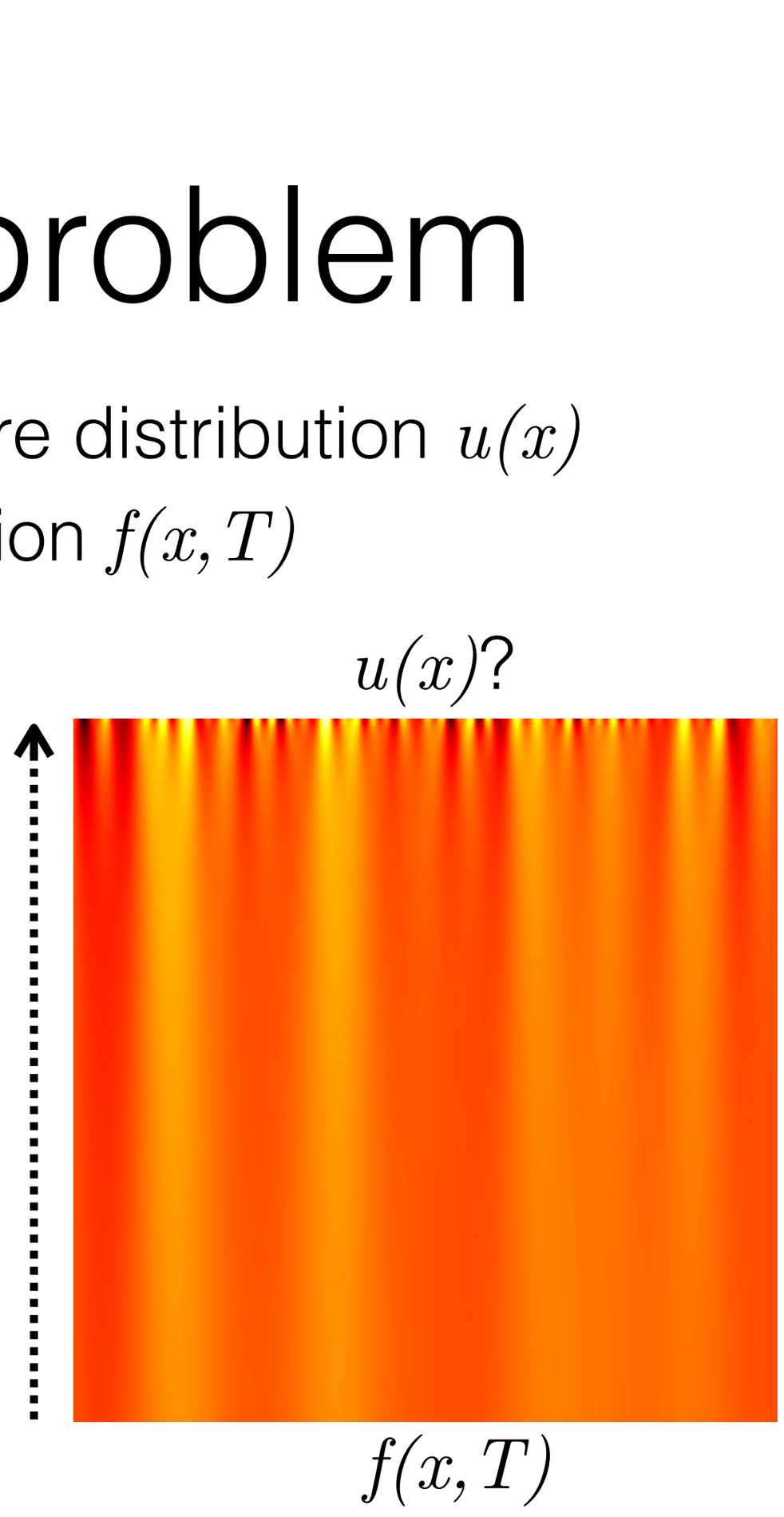
with thermal conductivity D and boundary conditions

$$f(x,0) = u(x)$$
$$f(0,t) = f(1,t) = 0$$



The inverse problem

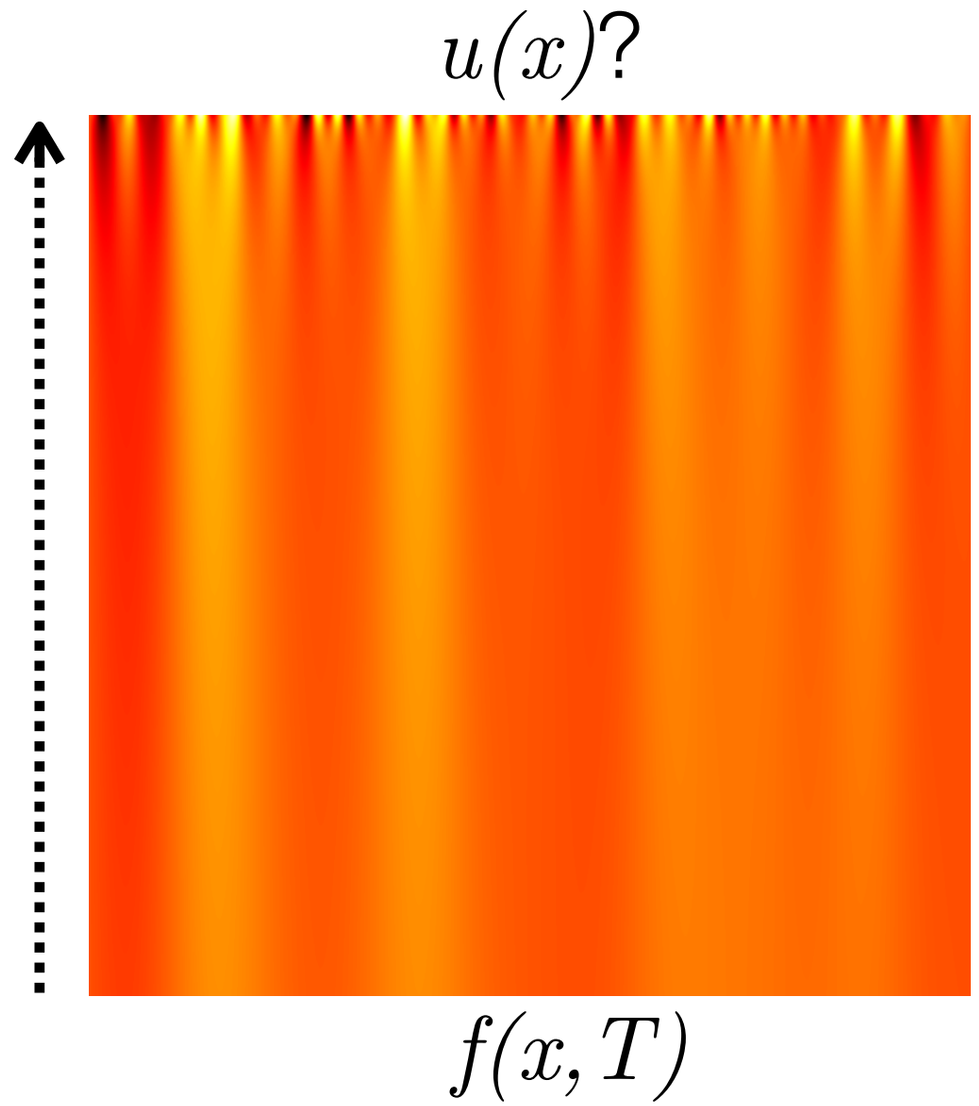
- Determine the initial temperature distribution $u(x)$ given the temperature distribution $f(x,T)$



The inverse problem

- Determine the initial temperature distribution $u(x)$ given the temperature distribution $f(x,T)$

- Errors propagate quickly

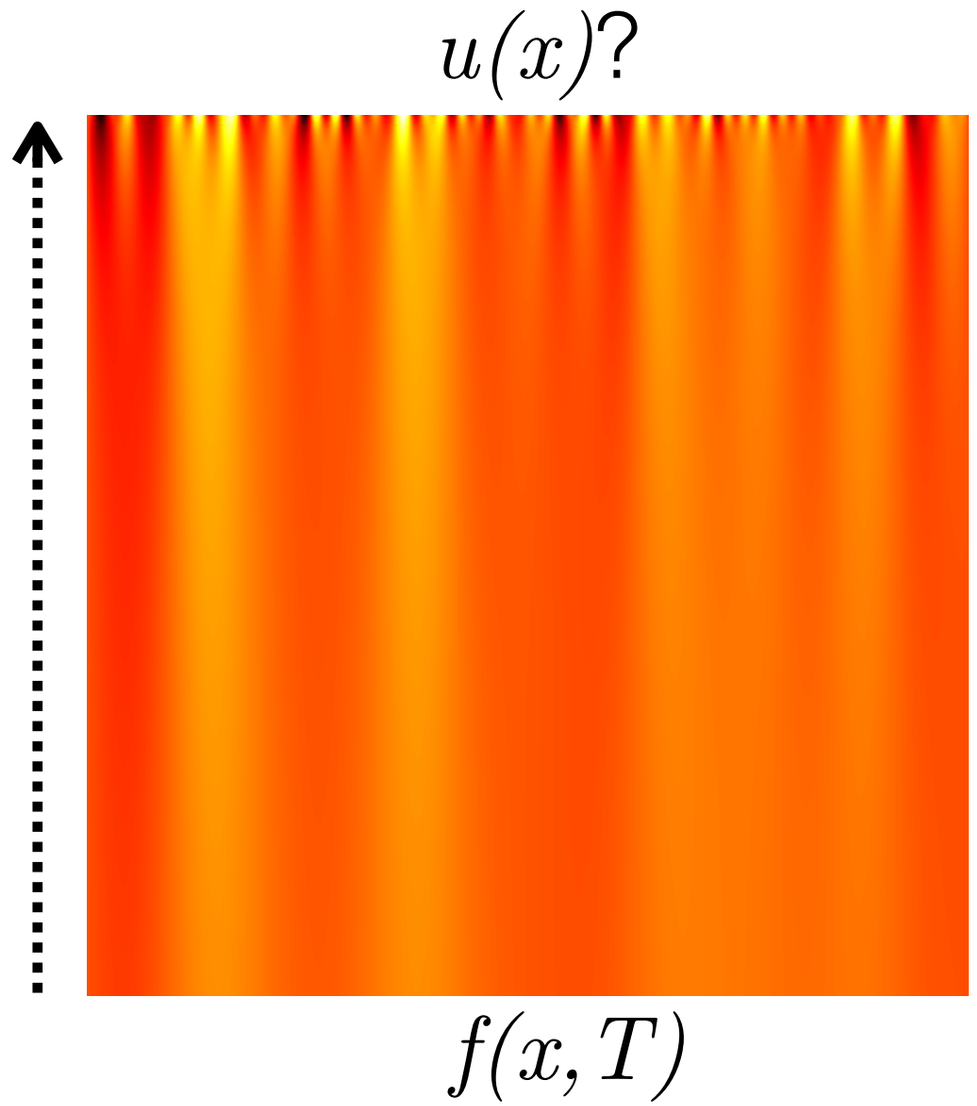


The inverse problem

- Determine the initial temperature distribution $u(x)$ given the temperature distribution $f(x,T)$

- Errors propagate quickly

- Thus, to determine $u(x)$, values of $f(x,T)$ must be stored with a very **high numerical accuracy**



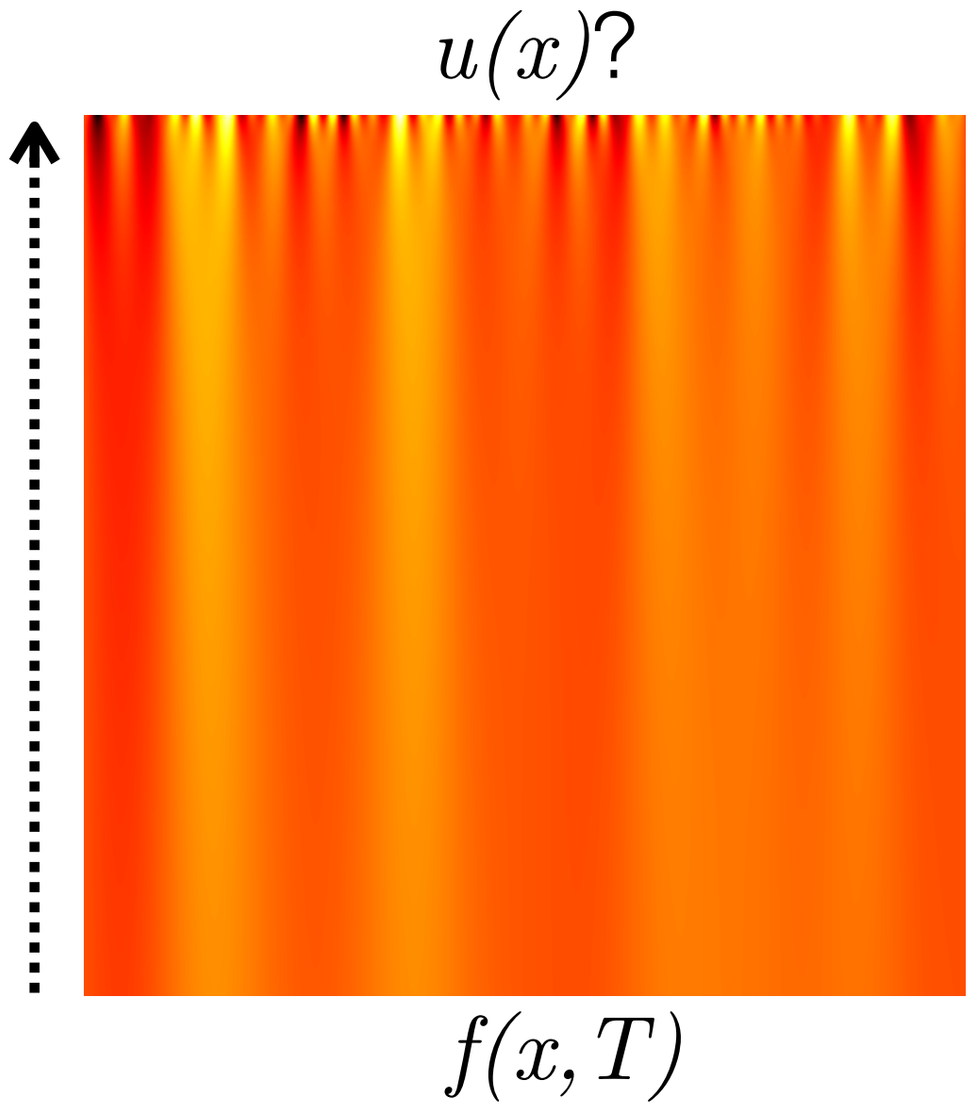
The inverse problem

- Determine the initial temperature distribution $u(x)$ given the temperature distribution $f(x,T)$

- Errors propagate quickly

- Thus, to determine $u(x)$, values of $f(x,T)$ must be stored with a very **high numerical accuracy**

- This poses a **limit** on what can be reconstructed



The inverse problem

- Exact recovery of $u(x)$ from $f(x,T)$ is not possible due to information loss (numerical resolution is finite)

The inverse problem

- Exact recovery of $u(x)$ from $f(x,T)$ is not possible due to information loss (numerical resolution is finite)
- Thus, many $u(x)$ yield the same distribution $f(x,T)$

The inverse problem

- Exact recovery of $u(x)$ from $f(x,T)$ is not possible due to information loss (numerical resolution is finite)

- Thus, many $u(x)$ yield the same distribution $f(x,T)$

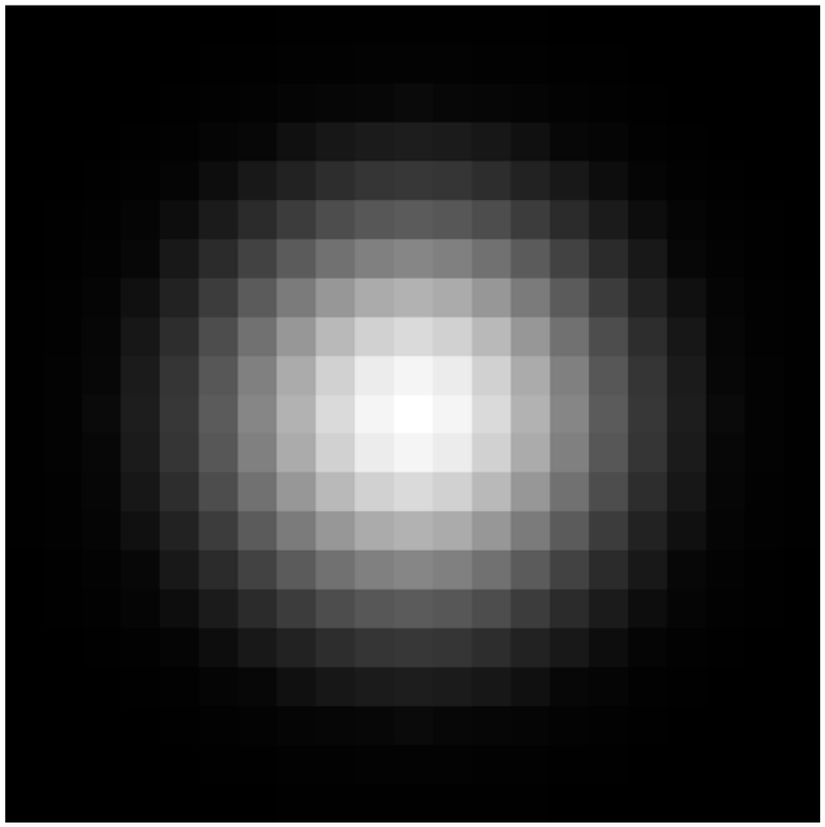
- Solving an **inverse problem** amounts to a **gain of information**

Another example: Deblurring

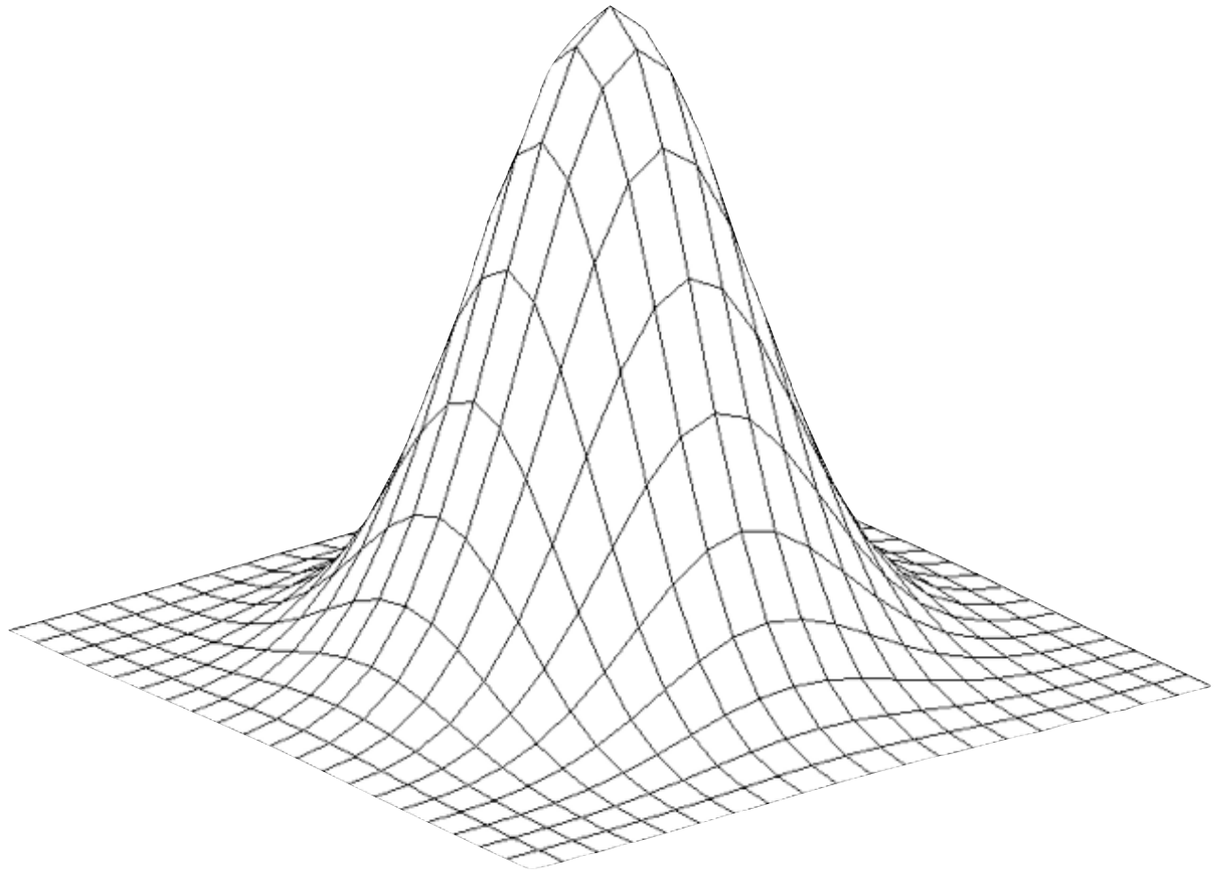
(unknown) object

(known) observation

Blur type



blur intensities (grayscale)



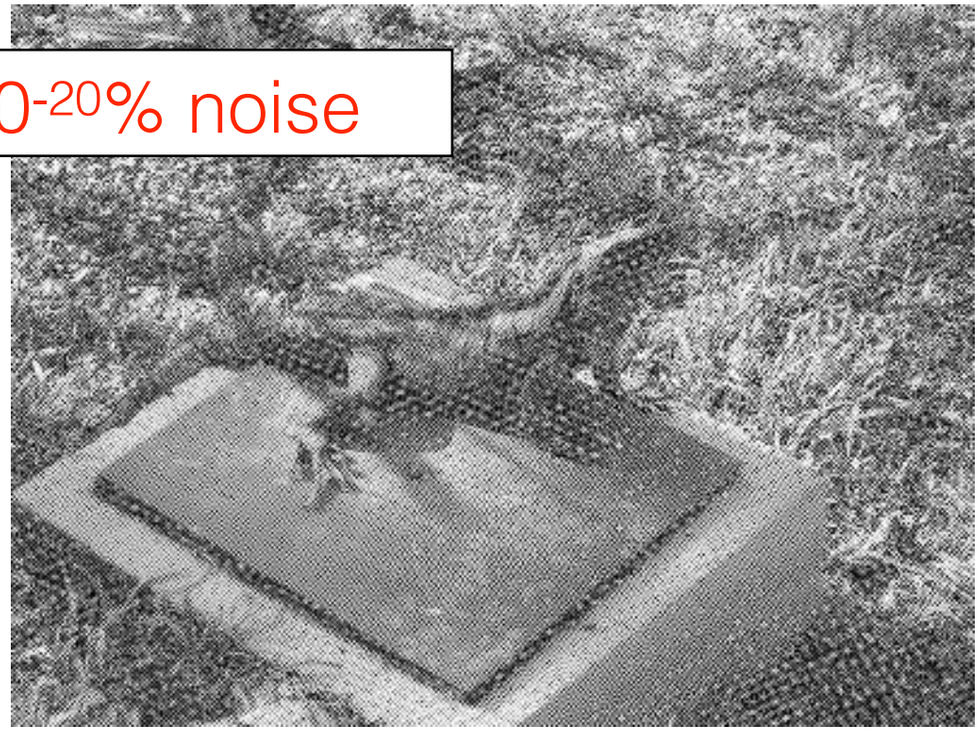
blur mesh plot

Object reconstructions

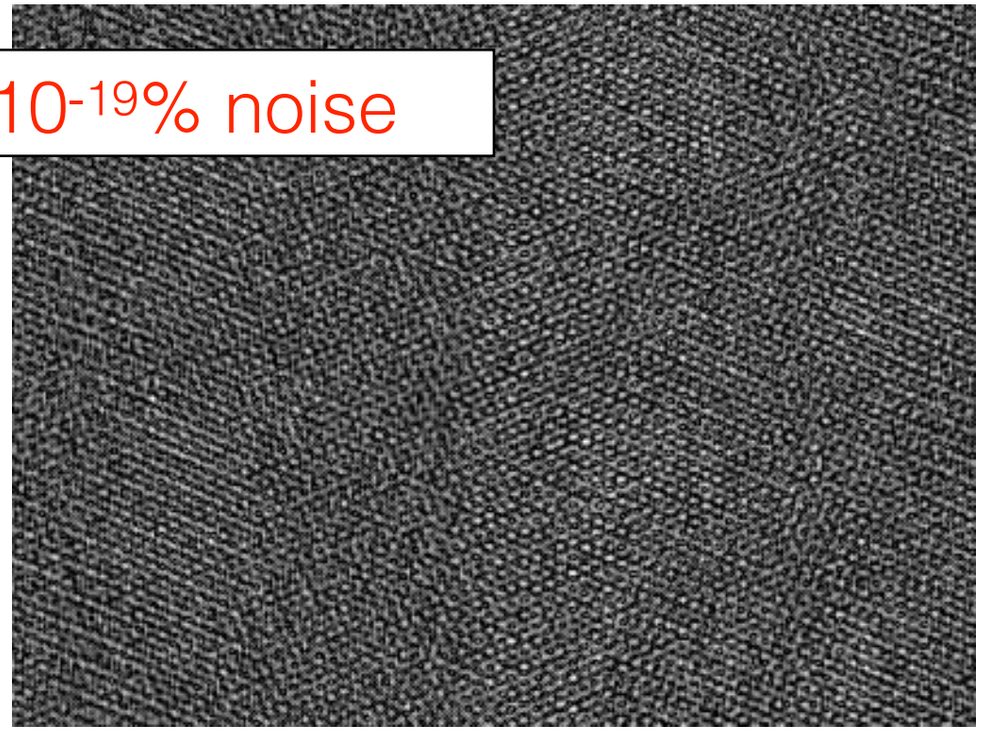
object

0% noise

10^{-20}% noise



10^{-19}% noise



Characterizing inverse problems

- **Definition (Hadamard)**: A problem is *ill-posed* if any of the following is true

Characterizing inverse problems

- **Definition (Hadamard)**: A problem is *ill-posed* if any of the following is true

(1) A solution (object) does not always exist

Characterizing inverse problems

- **Definition (Hadamard)**: A problem is *ill-posed* if any of the following is true

(1) A solution (object) does not always exist

(2) The solution (object) is not always unique

Characterizing inverse problems

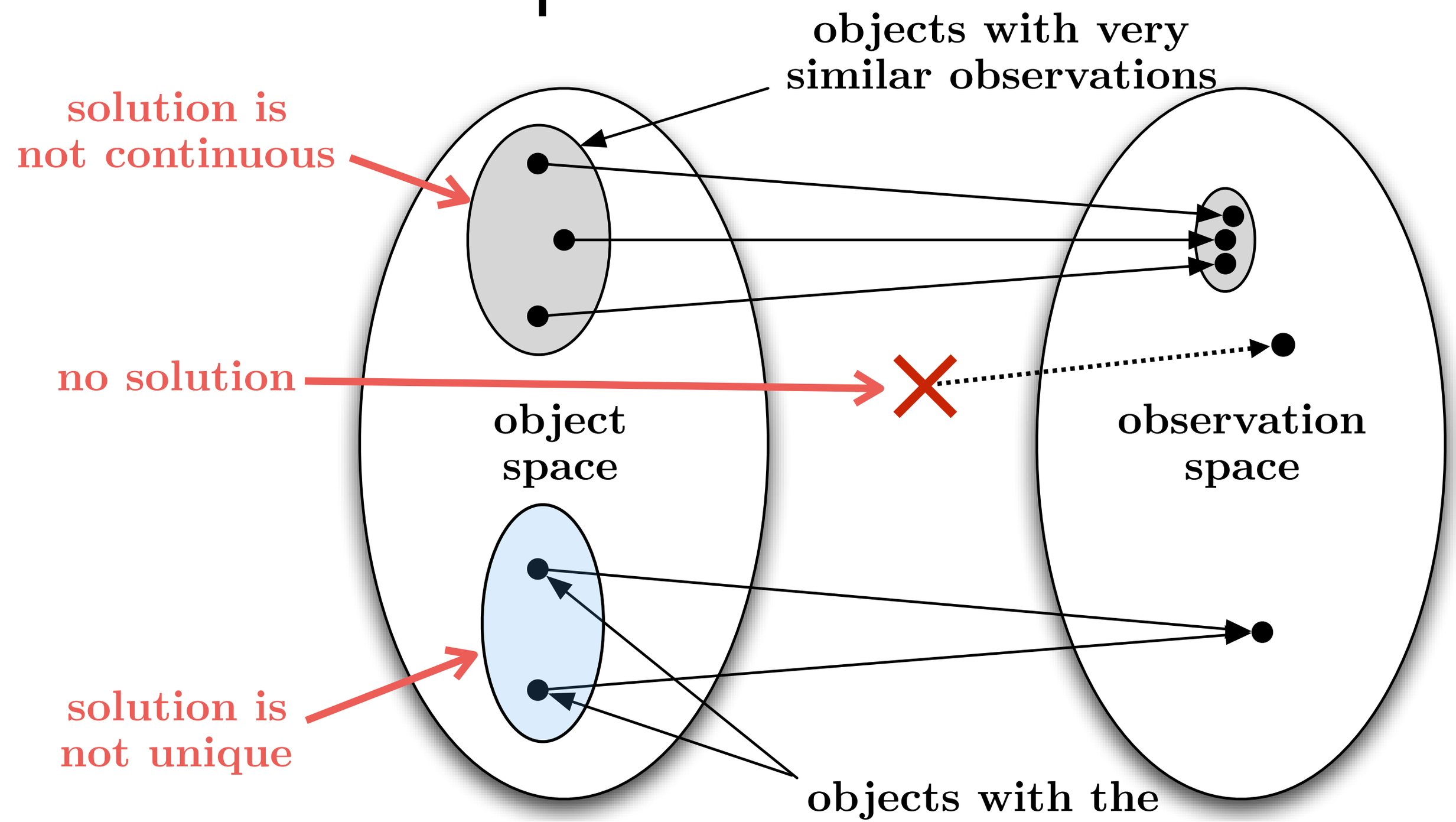
- **Definition (Hadamard)**: A problem is *ill-posed* if any of the following is true

(1) A solution (object) does not always exist

(2) The solution (object) is not always unique

(3) The solution (object) does not depend continuously on the data (observation)

Ill-posedness



Curing ill-posedness

- **General rule**
  - Define approximate solutions satisfying additional constraints (**a priori** information)

Curing ill-posedness

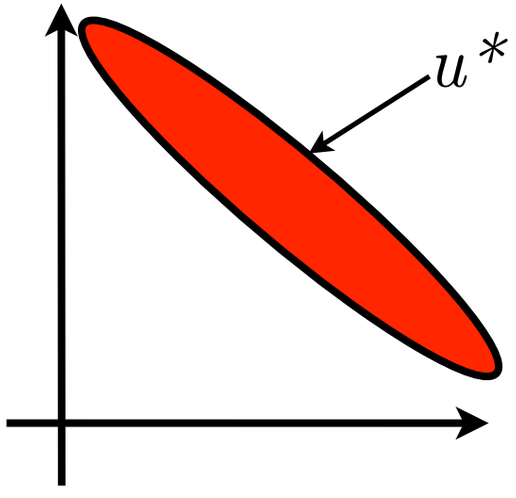
- **General rule**
  - Define approximate solutions satisfying additional constraints (**a priori** information)
  - This makes the mapping smoother and ensures that a solution always exists and is unique

Curing ill-posedness

- **General rule**
  - Define approximate solutions satisfying additional constraints (**a priori** information)
  - This makes the mapping smoother and ensures that a solution always exists and is unique
- **Examples**
  - Finite energy, positiveness, upper bounds, family of functions, smoothness, statistical properties

A-priori information and uniqueness

- Typically, the set of admissible solutions is too large



object space

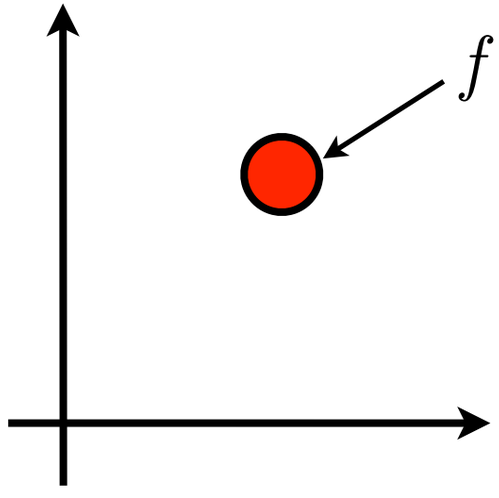


image space

A-priori information and uniqueness

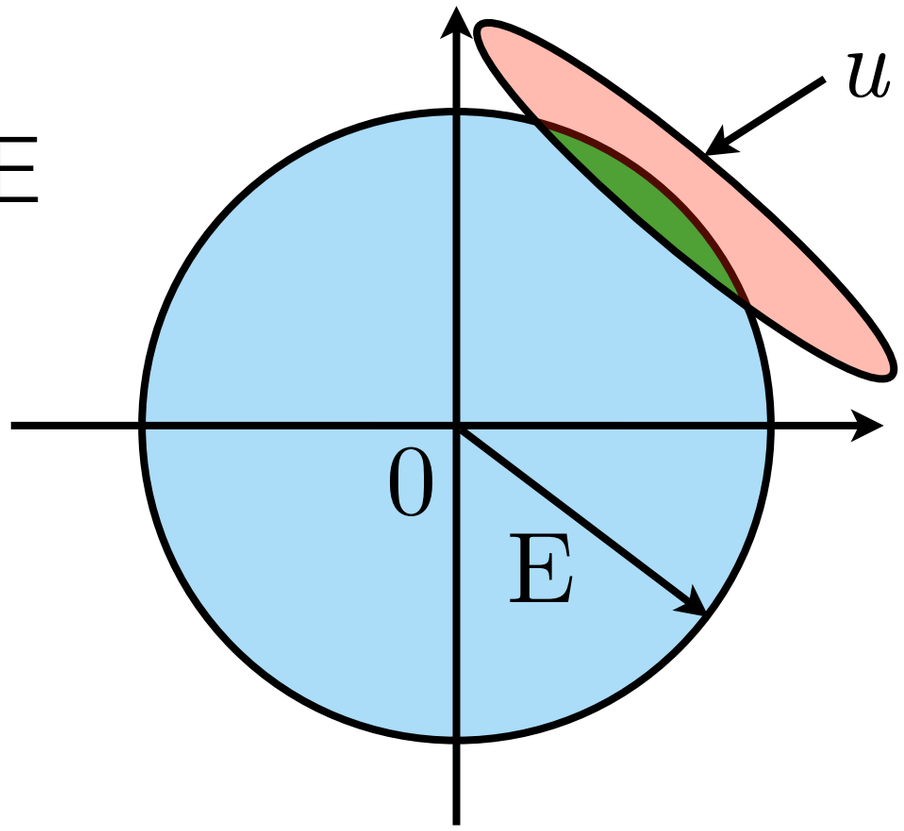
- The set of solutions can be reduced by introducing additional knowledge

- For example,
finite energy E

A-priori information and uniqueness

- The set of solutions can be reduced by introducing additional knowledge

- For example, finite energy E



object space

Regularization

- **Definition (A.N.Tikhonov)**
  - A **regularized solution** is one from a family of approximate solutions parametrized via a positive *regularization* parameter (typical math symbols ϵ, λ, μ)

- **Properties**
  1. When there is no noise, the family converges to the correct solution as the parameter goes to *0*
  2. When there is noise, the approximate solution can be obtained for a non zero parameter

Regularization

- **Example**: Suppose that the exact solution to an inverse problem is the function

$$u(x) = \frac{1}{x^2}$$

This function is not defined in $x=0$.

A regularized solution could be one from the approximate function family

$$u_\epsilon(x) = \frac{1}{x^2 + \epsilon}$$

This satisfies the two previous properties.

Regularization

- In general, given an optimization problem

$$\hat{u} = \arg\min_{u} E[u]$$

we can *regularize* it by changing it to

$$\hat{u}_\epsilon = \arg\min_{u} E[u] + \epsilon R[u]$$

where $R[u]$ is an energy term capturing the additional knowledge

Regularization

- **Example**

  - The original problem minimizes

$$\hat{u} = \arg\min_u E[u] \quad \text{with} \quad E[u] = |Au - f|^2$$

  - The solution is

$$\hat{u} = \left(A^\top A\right)^{-1} A^\top f$$

  - The inverse is not defined if A is not full rank.

Regularization

- **Example**

  - Introduce the regularized problem

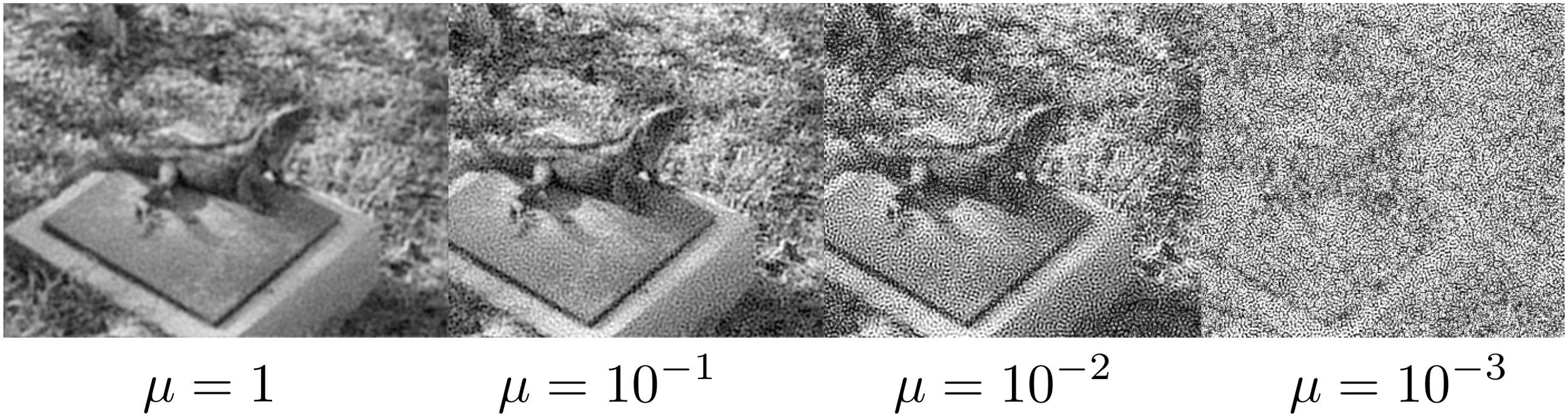
$$\hat{u}_\epsilon = \arg\min_u E[u] + \epsilon R[u] \quad \text{with} \quad \begin{array}{l} E[u] = |Au - f|^2 \\ R[u] = |u|^2 \end{array}$$

  - The solution is

$$\hat{u}_\epsilon = \left(A^\top A + \epsilon I\right)^{-1} A^\top f$$

  - This is always defined for all $\epsilon > 0$ and it becomes the original solution for $\epsilon = 0$

Example: Energy prior



$\mu = 1$ $\mu = 10^{-1}$ $\mu = 10^{-2}$ $\mu = 10^{-3}$

$\mu = 10$ $\mu = 10^{2}$ $\mu = 10^{3}$ $\mu = 10^{4}$

The Bayesian framework

- A more general framework that includes the energy minimization

The Bayesian framework

- A more general framework that includes the energy minimization

- *A priori* information can be given through a probability distribution

The Bayesian framework

- A more general framework that includes the energy minimization

- *A priori* information can be given through a probability distribution

- Formalizes previous experience or additional assumptions

Information about the object

- Denote the observation with f and the object with u
- The *prior* or *a priori* probability density function is $p(u)$
- This is our knowledge about the object
- Alternatively, we obtain it from the joint probability density function $p(u,f)$ via marginalization

$$p(u) = \int p(u, f)df$$

Bayes formula

- Relates the joint probability to the *prior*

$$p(u, f) = p(f|u)p(u)$$

via a conditional probability density function

A posteriori density function

- The *posterior* gives information about the object given the image

$$p(u|f) = \frac{p(u,f)}{p(f)} = \frac{p(f|u)p(u)}{\int p(f|\bar{u})p(\bar{u})d\bar{u}}$$

marginalization

Bayesian estimates

- The posterior provides all the information we can have about the object given the observations

- Most common point estimates

- **Conditional mean**

$$\tilde{u} = E[u|f] = \int up(u|f)du$$

- **Maximum a posteriori (MAP)**

$$\tilde{u} = \arg\max_u p(u|f)$$

Bayesian estimates

- The posterior provides all the information we can have about the object given the observations

- Most common point estimates

- **Conditional mean**

$$\tilde{u} = E[u|f] = \int up(u|f)du$$

- **Maximum a posteriori (MAP)**

$$\tilde{u} = \arg\max_u p(u|f)$$

Maximum a Posteriori

- We need to solve

$$\tilde{u} = \arg\max_{u} p(u|f)$$

Maximum a Posteriori

- We need to solve

$$\tilde{u} = \arg\max_u p(u|f)$$
$$= \arg\max_u p(f|u)p(u)$$

Maximum a Posteriori

- We need to solve

$$\begin{aligned}\tilde{u} &= \arg\max_u p(u|f) \\ &= \arg\max_u p(f|u)p(u) \\ &= \arg\max_u \log[p(f|u)p(u)]\end{aligned}$$

Maximum a Posteriori

- We need to solve

$$\begin{aligned}\tilde{u} &= \arg\max_u p(u|f) \\ &= \arg\max_u p(f|u)p(u) \\ &= \arg\max_u \log[p(f|u)p(u)] \\ &= \arg\max_u \log p(f|u) + \log p(u)\end{aligned}$$

Maximum a Posteriori

- We need to solve

$$\begin{aligned}
\tilde{u} &= \arg\max_u p(u|f) \\
&= \arg\max_u p(f|u)p(u) \\
&= \arg\max_u \log[p(f|u)p(u)] \\
&= \arg\max_u \log p(f|u) + \log p(u) \\
&= \arg\min_u -\log p(f|u) - \log p(u)
\end{aligned}$$

Maximum a Posteriori

- We have a minimization problem

$$\tilde{u} = \arg\min_u \underbrace{-\log p(f|u)}_{\text{image model}} \underbrace{-\log p(u)}_{\text{prior}}$$

- Compare to the previous regularization example: The prior takes the role of the regularization term

The optimization framework

- Most computer vision problems can be cast as an optimization task

- Inverse problems are often formulated in this framework

- Three main ingredients are needed

  - **Image model** (e.g., $f = k * u$ for deblurring)

  - **Observations** and parameters (e.g., f and k)

  - **Object prior** (e.g., $p(u)$)

The optimization framework

- The image model and observations form the so-called *data term*
- For example, suppose that the data error is Gaussian noise

$$p(f|u) \propto e^{-\frac{|f-k*u|^2}{2}}$$

then, we have

$$\log p(f|u) = -\frac{|f-k*u|^2}{2} + \text{constant}$$

The optimization framework

- The prior forms the so-called *regularization term*
- For example, suppose that the gradients are Gaussian random variables

$$p(u) \propto e^{-\lambda \frac{|\nabla u|^2}{2}}$$

then, we have

$$\log p(u) = -\lambda \frac{|\nabla u|^2}{2} + \text{constant}$$

The optimization framework

- The cost functional to be optimized combines the two terms (the formulation may vary)

$$\min_u |f - k * u|^2 + \lambda |\nabla u|^2$$

How do we find the solution?

- Now assume that a cost functional is provided

$$\tilde{u} = \arg\min_{u} -\log p(f|u) - \log p(u)$$

- To find the solution to the optimization problem we need to

How do we find the solution?

- Now assume that a cost functional is provided

$$\tilde{u} = \arg\min_{u} -\log p(f|u) - \log p(u)$$

- To find the solution to the optimization problem we need to
  - Define a numerical **representation** for the solution

How do we find the solution?

- Now assume that a cost functional is provided

$$\tilde{u} = \arg\min_{u} -\log p(f|u) - \log p(u)$$

- To find the solution to the optimization problem we need to
  - Define a numerical **representation** for the solution
  - Find a numerically viable formulation of the **optimality conditions**

Solution representation

- Problems are typically defined in the continuum

- Since computers can only represent and compute finite quantities, **discretization** in some domain is needed

Discretization

- Discretization is a truncation in some basis
- Approaches
  - Discretize energy, compute (exact) finite dimensional gradients, and solve discrete necessary conditions
  - Compute exact analytic form of gradients, discretize gradients, and solve discrete necessary conditions

Discretization

- Discretization is a truncation in some basis
- Approaches
  - Discretize energy, compute (exact) finite dimensional gradients, and solve discrete necessary conditions
  - Compute exact analytic form of gradients, discretize gradients, and solve discrete necessary conditions

Denoising example

- Denoising/Smoothing problem in 1D

$$\min_u |u - f|^2 + \frac{\lambda}{2}|\nabla u|^2$$

- We can discretize the data term $|u - f|^2$ as

$$\sum_{i=1}^{N}(u[i] - f[i])^2$$

- How do we discretize the gradient term $|\nabla u|^2$?

Discretizing derivatives: finite differences

- The analytic derivative is defined by

$$u'(x) = \lim_{\epsilon \mapsto 0} \frac{u(x+\epsilon) - u(x)}{\epsilon}$$

- The discrete approximation is obtained by using the smallest ϵ possible (the grid step is typically set to $\epsilon = 1$)

$$u'[x] \simeq \frac{u[x+\epsilon] - u[x]}{\epsilon}$$

Finite differences

- Forward difference (first order)

$$u'[x] \simeq \frac{u[x+\epsilon] - u[x]}{\epsilon}$$

- Backward difference (first order)

$$u'[x] \simeq \frac{u[x] - u[x-\epsilon]}{\epsilon}$$

- Central difference (first order)

$$u'[x] \simeq \frac{u[x+\epsilon] - u[x-\epsilon]}{2\epsilon}$$

Denoising example

- Discretize energy

$$\min_u \sum_{i=1}^{N} (u[i] - f[i])^2 + \frac{\lambda}{2} \sum_{i=1}^{N-1} (u[i+1] - u[i])^2$$

forward differences

Denoising example

- Given the discrete energy

$$\min_u \sum_{i=1}^{N} (u[i] - f[i])^2 + \frac{\lambda}{2} \sum_{i=1}^{N-1} (u[i+1] - u[i])^2$$

Denoising example

- Given the discrete energy

$$\min_u \sum_{i=1}^{N} (u[i] - f[i])^2 + \frac{\lambda}{2} \sum_{i=1}^{N-1} (u[i+1] - u[i])^2$$

check the boundaries

Denoising example

- Given the discrete energy

$$\min_u \sum_{i=1}^{N} (u[i] - f[i])^2 + \frac{\lambda}{2} \sum_{i=1}^{N-1} (u[i+1] - u[i])^2$$

- Compute the finite gradient $\forall i \in [2, N-1]$

$$\nabla_u E[i] = 2(u[i] - f[i]) + \lambda(2u[i] - u[i+1] - u[i-1])$$

- What is the gradient for $i{=}1$ and $i{=}N$?

Denoising example

- Given the discrete energy

$$\min_u \sum_{i=1}^{N} (u[i] - f[i])^2 + \frac{\lambda}{2} \sum_{i=1}^{N-1} (u[i+1] - u[i])^2$$

- Compute the finite gradient $\forall i \in [2, N-1]$

$$\nabla_u E[i] = 2(u[i] - f[i]) + \lambda(2u[i] - u[i+1] - u[i-1])$$

- What is the gradient for $i{=}1$ and $i{=}N$?

$$\nabla_u E[1] = 2(u[1] - f[1]) + \lambda(u[1] - u[2])$$
$$\nabla_u E[N] = 2(u[N] - f[N]) + \lambda(u[N] - u[N-1])$$

Denoising example

- Given the finite gradient

$$\nabla_u E[i] = 2(u[i] - f[i]) + \lambda(2u[i] - u[i+1] - u[i-1])$$

$$\forall i \in [2, N-1]$$

$$\nabla_u E[1] = 2(u[1] - f[1]) + \lambda(u[1] - u[2])$$

$$\nabla_u E[N] = 2(u[N] - f[N]) + \lambda(u[N] - u[N-1])$$

- Solve the necessary conditions

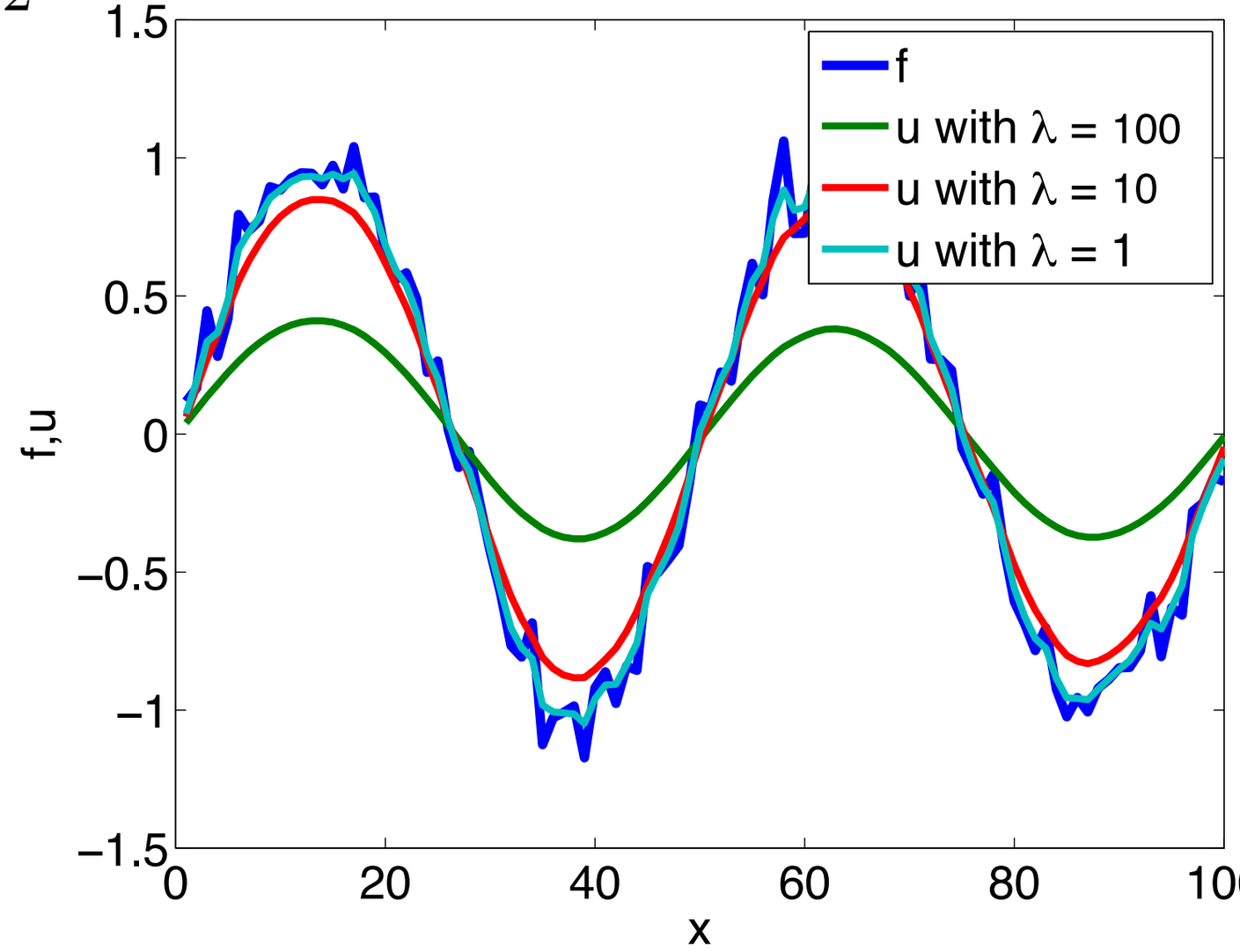
$$\nabla_u E = 0$$

Denoising example

- Denoising/Smoothing problem

$$\min_u |u - f|^2 + \frac{\lambda}{2}|\nabla u|^2$$

- Results with different regularization parameters



Solving the necessary conditions

- We need to solve the necessary conditions

$$\nabla_u E = 0$$

- If it is a linear system (e.g., denoising example), solve it directly

$$\nabla_u E[u] = Au - b = 0 \quad \Rightarrow \quad \tilde{u} = A^{-1}b$$

Solving the necessary conditions

- Solving $u = A^{-1}b$ may seem unfeasible when the number of entries of u is in the order of Millions

- However, notice that these matrices are extremely **sparse** and **diagonal**

- Memory and computationally efficient methods exist to solve these systems (also in Matlab)

Solving the necessary conditions

- The finite gradient is linear $\forall i \in [2, N-1]$

$$\nabla_u E[i] = 2(u[i] - f[i]) + \lambda(2u[i] - u[i+1] - u[i-1])$$
$$\nabla_u E[1] = 2(u[1] - f[1]) + \lambda(u[1] - u[2])$$
$$\nabla_u E[N] = 2(u[N] - f[N]) + \lambda(u[N] - u[N-1])$$

solve directly via $u = A^{-1}b$ where $b[i] = 2f[i]$ and

$$\forall i \in [2, N-1]$$
$$A[i,i] = 2 + 2\lambda$$
$$A[i,i+1] = -\lambda$$
$$A[i,i-1] = -\lambda$$

$$A[1,1] = 2 + \lambda$$
$$A[1,2] = -\lambda$$
$$A[N,N-1] = -\lambda$$
$$A[N,N] = 2 + \lambda$$

$$A[i,j] = 0 \quad \forall j \neq i, i-1, i+1$$

Solving the necessary conditions

- The finite gradient is linear $\forall i \in [2, N-1]$

$$\nabla_u E[i] = 2(u[i] - f[i]) + \lambda(2u[i] - u[i+1] - u[i-1])$$
$$\nabla_u E[1] = 2(u[1] - f[1]) + \lambda(u[1] - u[2])$$
$$\nabla_u E[N] = 2(u[N] - f[N]) + \lambda(u[N] - u[N-1])$$

solve directly via $u = A^{-1}b$ where $b[i] = 2f[i]$ and

$$\forall i \in [2, N-1]$$
$$A[i,i] = 2 + 2\lambda$$
$$A[i,i+1] = -\lambda$$
$$A[i,i-1] = -\lambda$$

$$A[1,1] = 2 + \lambda$$
$$A[1,2] = -\lambda$$
$$A[N,N-1] = -\lambda$$
$$A[N,N] = 2 + \lambda$$

$$A[i,j] = 0 \quad \forall j \neq i, i-1, i+1$$

Boundary conditions

- Consider the deblurring problem

$$\min_u |k * u - f|^2 + \frac{\lambda}{2}|\nabla u|^2$$

- In discrete form

$$\min_u \sum_{i=1}^{N} \left(\sum_{j=-W}^{W} k[j]u[i-j] - f[i] \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{N-1} (u[i+1] - u[i])^2$$

Boundary conditions

- Consider the deblurring problem

$$\min_u |k * u - f|^2 + \frac{\lambda}{2}|\nabla u|^2$$

- In discrete form

$$\min_u \sum_{i=1}^{N} \left(\sum_{j=-W}^{W} k[j]u[i-j] - f[i] \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{N-1} (u[i+1] - u[i])^2$$

- Necessary conditions use coordinates of u beyond its support

Boundary conditions

- **Neumann**: Specify the derivatives at the boundary

$$\frac{du}{dx}(y) = f(y) \qquad \forall y \in \partial\Omega$$

- **Dirichlet**: Specify the function at the boundary

$$u(y) = f(y) \qquad \forall y \in \partial\Omega$$

- **Cauchy**: Both Neumann and Dirichlet

Boundary assumptions

- Function is set to a constant (e.g., 0) — Dirichlet

$$u[y] = 0 \qquad \forall y \notin \Omega$$

- Function derivative is set to a constant (e.g., 0) — Neumann

$$u'[y] = 0 \qquad \forall y \notin \Omega$$

- Function is mirrored

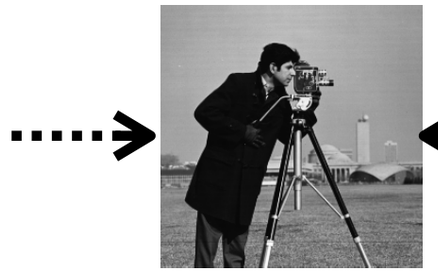
$$u[y] = u[\partial\Omega - y] \qquad \forall y \notin \Omega$$

- Function is periodic

$$u[y] = u[\partial\Omega + y] \qquad \forall y \notin \Omega$$

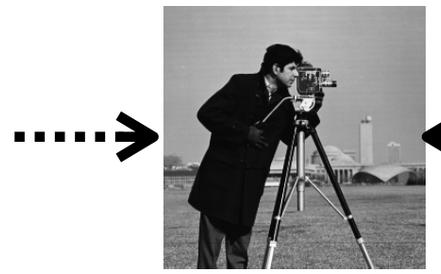
Boundary assumptions - 2D

- Images have a finite support



Boundary assumptions - 2D

- Images have a finite support

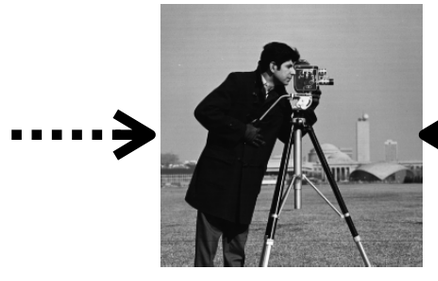


- How does the convolution operator behave at the image boundary?

$$f[i] = (k * u)[i] = \sum_{j=0}^{N} k[i-j]u[j]$$

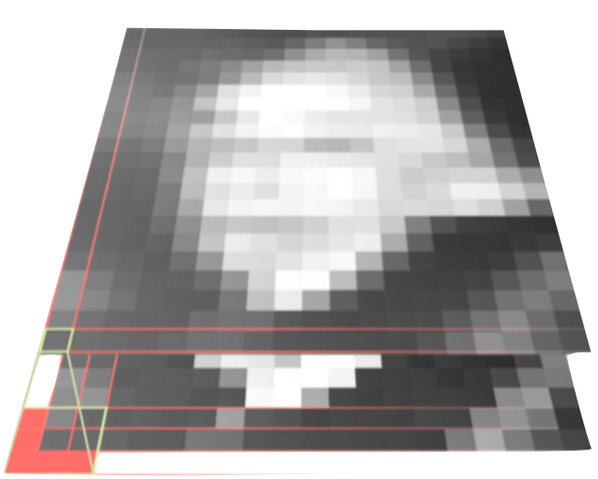
Boundary assumptions - 2D

- Images have a finite support



- How does the convolution operator behave at the image boundary?

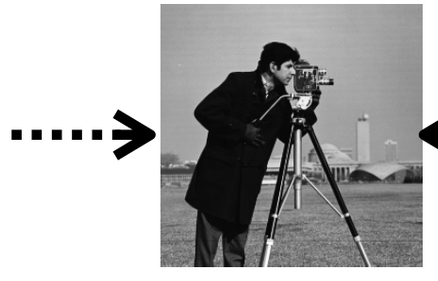
$$f[i] = (k * u)[i] = \sum_{j=0}^{N} k[i-j]u[j]$$



- One choice is that u and f have the **same** support

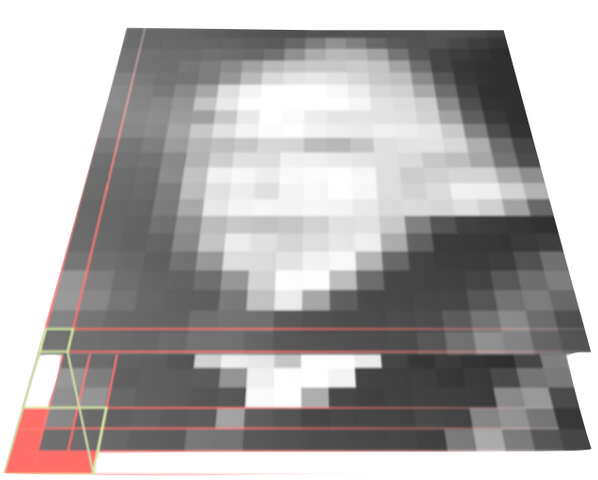
Boundary assumptions - 2D

- Images have a finite support



- How does the convolution operator behave at the image boundary?

$$f[i] = (k * u)[i] = \sum_{j=0}^{N} k[i-j]u[j]$$



- One choice is that u and f have the **same** support

- We need boundary assumptions for u

Boundary assumptions - 2D

- Boundary assumptions
  - zero (Dirichlet) $u[n+j]=0$
  - replicate
  - **periodic (FFT)**
  - symmetric
  - anti-reflective
- They all introduce artifacts
  - Smooth periodic extensions — Liu and Jia (2008)

Boundary assumptions - 2D

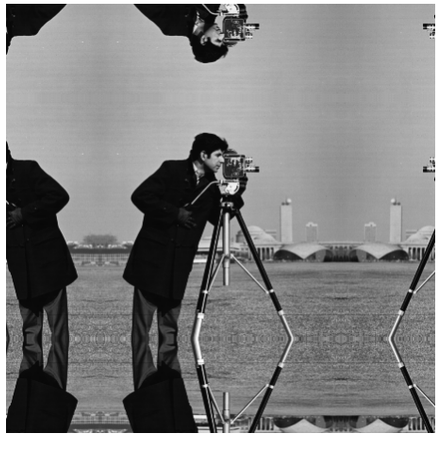
- Boundary assumptions
  - zero (Dirichlet) $u[n+j] = 0$
  - replicate $u[n+j] = u[n]$
  - **periodic (FFT)**
  - symmetric
  - anti-reflective
- They all introduce artifacts
  - Smooth periodic extensions — Liu and Jia (2008)

Boundary assumptions - 2D

- Boundary assumptions
  - zero (Dirichlet) $u[n+j] = 0$
  - replicate $u[n+j] = u[n]$
  - **periodic (FFT)** $u[n+j] = u[j]$
  - symmetric
  - anti-reflective
- They all introduce artifacts
  - Smooth periodic extensions — Liu and Jia (2008)

Boundary assumptions - 2D

- Boundary assumptions
  - zero (Dirichlet) $u[n+j] = 0$
  - replicate $u[n+j] = u[n]$
  - **periodic (FFT)** $u[n+j] = u[j]$
  - symmetric $u[n+j] = u[n+1-j]$
  - anti-reflective
- They all introduce artifacts
  - Smooth periodic extensions — Liu and Jia (2008)

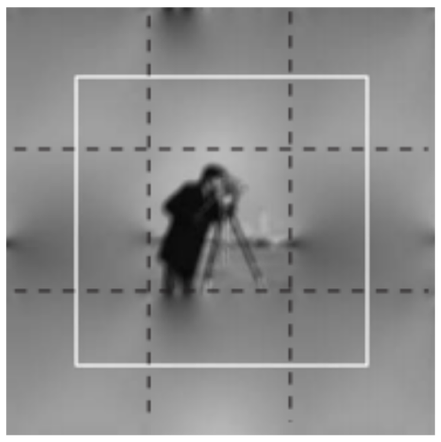


Boundary assumptions - 2D

- Boundary assumptions
  - zero (Dirichlet) $u[n+j] = 0$
  - replicate $u[n+j] = u[n]$
  - **periodic (FFT)** $u[n+j] = u[j]$
  - symmetric $u[n+j] = u[n+1-j]$
  - anti-reflective $u[n+j] = 2u[n] - u[n-j]$
- They all introduce artifacts
  - Smooth periodic extensions — Liu and Jia (2008)

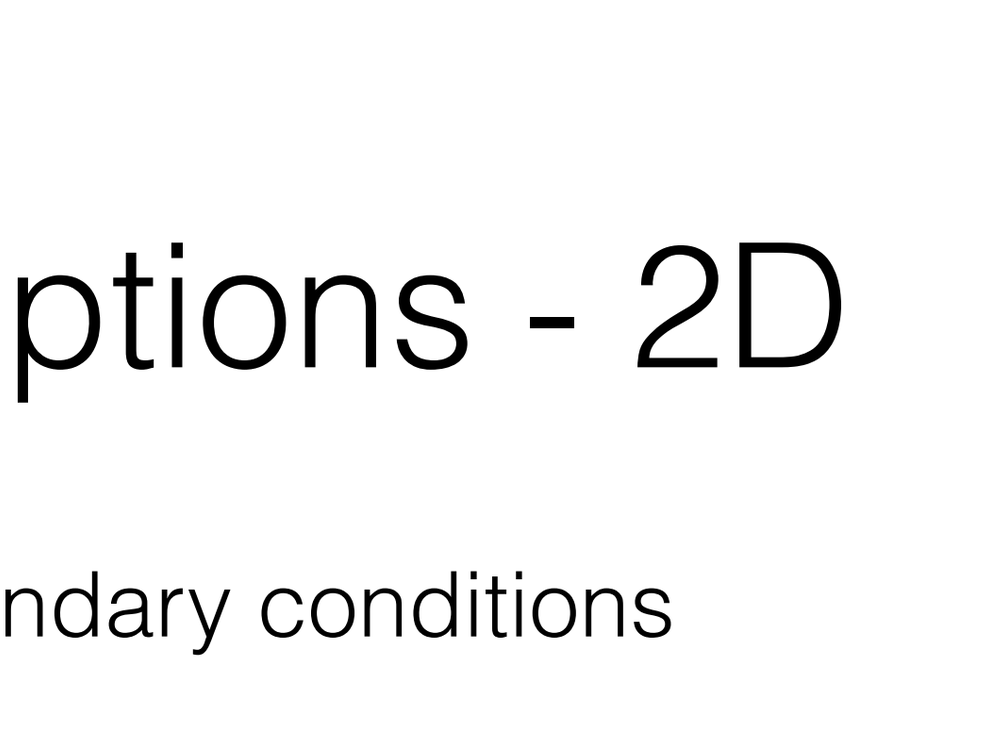
Boundary assumptions - 2D

- Boundary assumptions
  - zero (Dirichlet) $u[n+j] = 0$
  - replicate $u[n+j] = u[n]$
  - **periodic (FFT)** $u[n+j] = u[j]$
  - symmetric $u[n+j] = u[n+1-j]$
  - anti-reflective $u[n+j] = 2u[n] - u[n-j]$
- They all introduce artifacts
  - Smooth periodic extensions — Liu and Jia (2008)



Boundary assumptions - 2D

- Another option is to **avoid** boundary conditions
- The supports of u and f are **different**
  - No artifacts
  - No FFT
  - `valid` convolution in Matlab



2D example

- Total Variation (isotropic, continuum)

$$\|\nabla u\|_2 = \iint \sqrt{u_{x_1}^2(x_1, x_2) + u_{x_2}^2(x_1, x_2)} dx_1 dx_2$$

- Total Variation (isotropic, discrete)

$$\|\nabla u\|_2 = \sum_{i,j} \sqrt{u_{x_1}^2[i, j] + u_{x_2}^2[i, j]}$$

Total Variation

- Energy discretization (forward difference scheme)

$$\|\nabla u\|_2 \simeq \sum_{i,j} \sqrt{(u[i+1,j]-u[i,j])^2 + (u[i,j+1]-u[i,j])^2}$$

what terms in the sum depend on $u[i,j]$?

Total Variation

- Exact gradient of the discretized energy (away from the boundaries)

$$\frac{\partial \|\nabla u\|_2}{\partial u[i,j]} = \frac{\partial \tau[i,j]}{\partial u[i,j]} + \frac{\partial \tau[i-1,j]}{\partial u[i,j]} + \frac{\partial \tau[i,j-1]}{\partial u[i,j]}$$

with

$$\tau[i,j] \doteq \sqrt{(u[i+1,j]-u[i,j])^2 + (u[i,j+1]-u[i,j])^2}$$

Total Variation

- Exact gradient of the discretized energy

$$\frac{\partial \tau[i,j]}{\partial u[i,j]} = \frac{2u[i,j] - u[i+1,j] - u[i,j+1]}{\tau[i,j]}$$

non linear

The other terms follow in a similar manner (left as exercise)

Total Variation numerical details

- Division by zero

  - When $\tau[i,j] = 0$ the gradient cannot be computed

    - Approximate the energy with

$$\|\nabla u\|_2 \simeq \sum_{i,j} \sqrt{(u[i+1,j] - u[i,j])^2 + (u[i,j+1] - u[i,j])^2 + \delta}$$

small positive value

Total Variation numerical details

- Boundary conditions

  - The gradient formula at the boundary requires access to out-of-boundary values of u

  - Make a choice among the several possible boundary assumptions

Solving the necessary conditions

- In general the gradient is **not linear** in u

- In this case one can solve the necessary conditions iteratively via

  - **Gradient descent**

  - **Iterative linearization**

Gradient descent

- Gradient descent uses the following basic iteration

$$u^{t+1} = u^t - \epsilon \nabla E[u^t]$$

where ϵ is a positive constant, which we call the *step magnitude*

Gradient descent
Quick look

- Observations about $u^{t+1} = u^t - \epsilon \nabla E[u^t]$

  - It stops when the gradient is zero

  - It changes entries of u that correspond to a large gradient

  - Since the gradient corresponds to the direction of largest increase of the energy E, a gradient step aims to reduce the energy E

Gradient descent
An interpretation

- Necessary conditions for a minimum of E

$$\nabla_u E = 0$$

by using its first order Taylor expansion

$$0 \simeq \nabla E[u^{t+1}] = \nabla E[u^t] + HE[u^t](u^{t+1} - u^t)$$

and an upper bound on the Hessian (assume positive definite)

$$0 \simeq \nabla E[u^{t+1}] = \nabla E[u^t] + \frac{1}{\epsilon}(u^{t+1} - u^t)$$

Gradient descent Properties

- A proof that gradient descent minimizes the cost function E requires
  - Lipschitz continuity
  - A non zero gradient
  - A small enough step magnitude

Denoising example

- Gradient descent algorithm

  - Initialize $u = f$
  - Loop until gradient is small enough

compute gradient

$$\nabla_u E[1] = 2(u[1] - f[1]) + \lambda(u[1] - u[2])$$
$$\nabla_u E[N] = 2(u[N] - f[N]) + \lambda(u[N] - u[N-1])$$
$$\nabla_u E[i] = 2(u[i] - f[i]) + \lambda(2u[i] - u[i+1] - u[i-1]) \qquad \forall i \in [2, N-1]$$

update solution

$$u[i] \leftarrow u[i] - \epsilon \nabla_u E[i] \qquad \forall i \in [1, N]$$

Iterative linearization

- Write the Taylor expansion of the gradient $\nabla E[u]$ around the current estimate u^t and discard terms of order higher than one

$$\nabla E[u] = \nabla E[u^t] + HE[u^t]\left(u - u^t\right) + O(\left|u - u^t\right|^2)$$

Iterative linearization

- Write the Taylor expansion of the gradient $\nabla E[u]$ around the current estimate u^t and discard terms of order higher than one

$$\nabla E[u] = \nabla E[u^t] + HE[u^t]\left(u - u^t\right) + O(\left|u - u^t\right|^2)$$

Iterative linearization

- Write the Taylor expansion of the gradient $\nabla E[u]$ around the current estimate u^t and discard terms of order higher than one

$$\nabla E[u] = \nabla E[u^t] + HE[u^t]\left(u - u^t\right) + O(\left|u - u^t\right|^2)$$

- Define next update u^{t+1} by setting the gradient to zero

$$\nabla E[u^{t+1}] = \nabla E[u^t] + HE[u^t]\left(u^{t+1} - u^t\right) = 0$$

Iterative linearization

- The locally linearized gradient

$$\nabla E[u^{t+1}] = \nabla E[u^t] + HE[u^t]\left(u^{t+1} - u^t\right) = 0$$

is a linear system $Au^{t+1} = b$ where

$$A = HE[u^t]$$

$$b = HE[u^t]u^t - \nabla E[u^t]$$

Iterative linearization algorithm

- Loop until changes are negligible

    1. Locally linearize gradient (around u^t)
    2. Solve linear system $Au^{t+1} = b$

- Notice: **no gradient step**

- Linear systems solvers and speedups:
  Matrix factorization, Jacobi, Gauss-Seidel,
  Successive Over-Relaxations, Multigrid Methods

Iterative linearization algorithm

- Loop until changes are negligible

    1. Locally linearize gradient (around u^t)
    2. Solve linear system $Au^{t+1} = b$

- Notice: **no gradient step**

- Linear systems solvers and speedups:
  Matrix factorization, Jacobi, Gauss-Seidel,
  Successive Over-Relaxations, Multigrid Methods

Iterative linearization algorithm

- Loop until changes are negligible

    1. Locally linearize gradient (around u^t)
    2. Solve linear system $Au^{t+1} = b$

- Notice: **no gradient step**

- Linear systems solvers and speedups: Matrix factorization, Jacobi, Gauss-Seidel, Successive Over-Relaxations, Multigrid Methods

Example

- Let us consider 1D total variation denoising

$$\min_u |u - f|^2 + \lambda |\nabla u|$$

- As before we discretize the data term $|u - f|^2$ as

$$\sum_{i=1}^{N} (u[i] - f[i])^2$$

- What about the regularization (total variation) term?

Example

- The 1D total variation term can be approximated via

$$|\nabla u| \simeq \sum_{i=1}^{N-1} \sqrt{(u[i+1] - u[i])^2 + \epsilon}$$

  for a small $\epsilon > 0$

- The analytic gradient is then $\forall i \in [2, N-1]$

$$\nabla E[i] = 2(u[i] - f[i]) + \lambda \left(\frac{u[i] - u[i+1]}{\sqrt{(u[i+1] - u[i])^2 + \epsilon}} + \frac{u[i] - u[i-1]}{\sqrt{(u[i] - u[i-1])^2 + \epsilon}} \right)$$

Example

- The gradient is not a linear system
- One can apply iterative linearization $\forall i \in [2, N-1]$

$$\nabla E[i] = 2(u[i] - f[i])$$
$$+ \lambda \left(\frac{u[i] - u[i+1]}{\sqrt{(u[i+1] - u[i])^2 + \epsilon}} + \frac{u[i] - u[i-1]}{\sqrt{(u[i] - u[i-1])^2 + \epsilon}} \right)$$

and compute the gradient of the above (vectorial) function with respect to u

Solving linear systems

- We can solve linear systems $Au = b$ by explicitly computing the inverse of the matrix A

- This is unfeasible when A is large

- This might still be unfeasible even if we use the singular value decomposition $A = U\Sigma V^\top$ and

$$u = V\left(\Sigma^{-1}\right)^\top U^\top b$$

Solving linear systems

- How can we solve linear systems $Au = b$ when the matrix A is very large?

- We can use iterative methods such as Gauss-Seidel and Successive Over-Relaxation (SOR)

Gauss iteration

- We are interested in solving the linear system

$$Au = b$$

- Suppose that A is **diagonal**

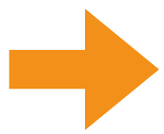
- We can solve element-wise

A_{11}	0	0	0
0	A_{22}	0	0
0	0	A_{33}	0
…	…	…	…

u_1
u_2
u_3
…

$=$

b_1
b_2
b_3
…



$u_1 = b_1 / A_{11}$
$u_2 = b_2 / A_{22}$
$u_3 = b_3 / A_{33}$
…

Gauss iteration

- We are interested in solving the linear system $Au = b$
- Decompose A as $A = D + N$
  where D is diagonal and N is zero on the diagonal
- Then define the following iterative solver

$$
\begin{aligned}
(D + N)u &= b \\
Du + Nu &= b \\
Du^{t+1} + Nu^{t} &= b \\
Du^{t+1} &= b - Nu^{t}
\end{aligned}
$$

Gauss iteration

- We are interested in solving the linear system $Au = b$
- Decompose A as $A = D + N$
  where D is diagonal and N is zero on the diagonal

- Then define the following iterative solver

$$(D + N)u = b$$

$$Du + Nu = b$$

$$Du^{t+1} + Nu^t = b$$

$$Du^{t+1} = b - Nu^t$$

element-wise solvable

Gauss iteration algorithm

- Given u^t
- Iterate for each i=1,…,N

$$u^{t+1}[i] = \frac{b[i] - \sum_{j \neq i} A[i,j] u^t[j]}{A[i,i]}$$

Gauss-Seidel

- Is an iterative method
- Exploits inversion of matrices in triangular form
- Works only for **diagonally dominant** or **symmetric** and **positive definite** matrices
    - This is typically the case in our optimization problems due to regularization

Lower triangular matrices

- We are interested in solving the linear system

$$Au = b$$

- Suppose that A is **lower triangular**

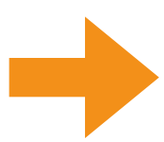
- We can use (Gauss) back substitution

A_{11}	0	0	0
A_{21}	A_{22}	0	0
A_{31}	A_{32}	A_{33}	0
…	…	…	…

u_1
u_2
u_3
…

$=$

b_1
b_2
b_3
…



execution order

$u_1 = b_1 / A_{11}$
$u_2 = (b_2 - A_{21}u_1) / A_{22}$
$u_3 = (b_3 - A_{31}u_1 - A_{32}u_2) / A_{33}$
…

Gauss-Seidel decomposition

- We are interested in solving the linear system $Au = b$
- Decompose A as $A = L + U,$ where L is lower triangular and U strictly upper triangular
- Then define the following solver

$$
\begin{aligned}
(L + U)u &= b \\
Lu + Uu &= b \\
Lu^{t+1} + Uu^{t} &= b \\
Lu^{t+1} &= b - Uu^{t}
\end{aligned}
$$

Gauss-Seidel decomposition

- We are interested in solving the linear system $Au = b$
- Decompose A as $A = L + U,$ where L is lower triangular and U strictly upper triangular
- Then define the following solver

$$(L + U)u = b$$

$$Lu + Uu = b$$

$$Lu^{t+1} + Uu^t = b$$

$$Lu^{t+1} = b - Uu^t$$

use back substitution

Gauss-Seidel iteration algorithm

- Improvement over Gauss iteration

$$u^{t+1}[i] = \frac{b[i] - \sum_{j=1}^{i-1} A[i,j]u^{t+1}[j] - \sum_{j=i+1}^{N} A[i,j]u^{t}[j]}{A[i,i]}$$

because we can use current updated values

Gauss-Seidel iteration algorithm

- Improvement over Gauss iteration

$$u^{t+1}[i] = \frac{b[i] - \sum_{j=1}^{i-1} A[i,j]u^{t+1}[j] - \sum_{j=i+1}^{N} A[i,j]u^{t}[j]}{A[i,i]}$$

because we can use current updated values

Successive over-relaxations

- Decompose A into three matrices

$$A = D + L + U$$

where D is diagonal

L strictly lower triangular

U strictly upper triangular

Successive over-relaxations

- Given the decomposition of A

$$A = D + L + U$$

- Write the linear system $Au = b$ as

$$(D + \omega L)u = \omega b - (\omega U + (\omega - 1)D)u$$

where ω can be arbitrary

Successive over-relaxations

- Given the new form of the linearized system

$$(D + \omega L)u = \omega b - (\omega U + (\omega - 1)D)u$$

set the right hand side to the current u values and consider only the left hand side for the update

Successive over-relaxations

- Given the new form of the linearized system

$$(D + \omega L)u = \omega b - (\omega U + (\omega - 1)D)u^t$$

set the right hand side to the current u values and consider only the left hand side for the update

Successive over-relaxations

- Given the new form of the linearized system

$$(D + \omega L)u^{t \pm 1} \omega b - (\omega U + (\omega - 1)D)u^t$$

set the right hand side to the current u values and consider only the left hand side for the update

Successive over-relaxations

- Given the new form of the linearized system

$$(D + \omega L)u^{t \pm 1} \omega b - (\omega U + (\omega - 1)D)u^t$$

  set the right hand side to the current u values and consider only the left hand side for the update

- This yields

$$u^{t+1} = (D + \omega L)^{-1} \left(\omega b - [\omega U + (\omega - 1)D]u^t\right)$$

Successive over-relaxations

- Given the new form of the linearized system

$$(D + \omega L)u^{t \pm 1} \omega b - (\omega U + (\omega - 1)D)u^t$$

  set the right hand side to the current u values and consider only the left hand side for the update

- This yields

$$u^{t+1} = (D + \omega L)^{-1} \left(\omega b - [\omega U + (\omega - 1)D]u^t\right)$$

use back substitution

Successive over-relaxations algorithm

- The numerical algorithm to solve

$$u^{t+1} = (D + \omega L)^{-1} \left(\omega b - [\omega U + (\omega - 1)D]u^t\right)$$

is, for i=1,...,N with $w \in (1, 2)$

$$u^{t+1}[i] = (1 - \omega)u^t[i] + \frac{\omega}{A[i,i]} \left(b[i] - \sum_{j=1}^{i-1} A[i,j]u^{t+1}[j] - \sum_{j=i+1}^{N} A[i,j]u^t[j] \right)$$