Practical Course: GPU Programming in Computer Vision

Mathematics 4: Variational Methods

Björn Häfner, Robert Maier, David Schubert

Technische Universität München
Department of Informatics
Computer Vision Group

Summer Semester 2018
September 17 - October 15

Outline

Outline

Variational methods

Energy minimization
An established approach to model numerous computer vision problems.

Energy
Every possible candidate solution u is assigned an *energy* $E(u)$.
Idea: $E(u)$ measures the *costs* of u: The smaller the costs the better the solution.

Minimizers
Candidates u with *least* energy are considered solutions to the problem.

Advantages:

- Clear mathematical correspondence between input data and result
- Extensive mathematical theory, optimality conditions
- Can describe sophisticated problems with only a few parameters
- Lots of algorithms to compute the minimizers

Variational methods

Energy minimization
An established approach to model numerous computer vision problems.

Energy
Every possible candidate solution u is assigned an *energy* $E(u)$.
Idea: $E(u)$ measures the *costs* of u: The smaller the costs the better the solution.

Minimizers
Candidates u with *least* energy are considered solutions to the problem.

Advantages:

- Clear mathematical correspondence between input data and result
- Extensive mathematical theory, optimality conditions
- Can describe sophisticated problems with only a few parameters
- Lots of algorithms to compute the minimizers

Variational methods

Energy minimization
An established approach to model numerous computer vision problems.

Energy
Every possible candidate solution u is assigned an *energy* $E(u)$.
Idea: $E(u)$ measures the *costs* of u: The smaller the costs the better the solution.

Minimizers
Candidates u with *least* energy are considered solutions to the problem.

Advantages:

- Clear mathematical correspondence between input data and result
- Extensive mathematical theory, optimality conditions
- Can describe sophisticated problems with only a few parameters
- Lots of algorithms to compute the minimizers

Variational methods

Energy minimization
An established approach to model numerous computer vision problems.

Energy
Every possible candidate solution u is assigned an *energy* $E(u)$.
Idea: $E(u)$ measures the *costs* of u: The smaller the costs the better the solution.

Minimizers
Candidates u with *least* energy are considered solutions to the problem.

Advantages:

- Clear mathematical correspondence between input data and result
- Extensive mathematical theory, optimality conditions
- Can describe sophisticated problems with only a few parameters
- Lots of algorithms to compute the minimizers

Variational methods

Energy minimization
An established approach to model numerous computer vision problems.

Energy
Every possible candidate solution u is assigned an *energy* $E(u)$.
Idea: $E(u)$ measures the *costs* of u: The smaller the costs the better the solution.

Minimizers
Candidates u with *least* energy are considered solutions to the problem.

Advantages:

- Clear mathematical correspondence between input data and result
- Extensive mathematical theory, optimality conditions
- Can describe sophisticated problems with only a few parameters
- Lots of algorithms to compute the minimizers

Variational methods

Energy minimization
An established approach to model numerous computer vision problems.

Energy
Every possible candidate solution u is assigned an *energy* $E(u)$.
Idea: $E(u)$ measures the *costs* of u: The smaller the costs the better the solution.

Minimizers
Candidates u with *least* energy are considered solutions to the problem.

Advantages:

- Clear mathematical correspondence between input data and result
- Extensive mathematical theory, optimality conditions
- Can describe sophisticated problems with only a few parameters
- Lots of algorithms to compute the minimizers

Variational methods

Energy minimization
An established approach to model numerous computer vision problems.

Energy
Every possible candidate solution u is assigned an *energy* $E(u)$.
Idea: $E(u)$ measures the *costs* of u: The smaller the costs the better the solution.

Minimizers
Candidates u with *least* energy are considered solutions to the problem.

Advantages:

- Clear mathematical correspondence between input data and result
- Extensive mathematical theory, optimality conditions
- Can describe sophisticated problems with only a few parameters
- Lots of algorithms to compute the minimizers

Variational Methods

Typical form

$$E(u) = D(u) + R(u)$$

- **Data term** $D(u)$ measures how well the solution u fits input data.
- **Regularizer** $R(u)$ enforces regularity and smoothness of u.

Minimizing E will give a solution u which fits to the inputs and is smooth!

Variational Methods

Typical form

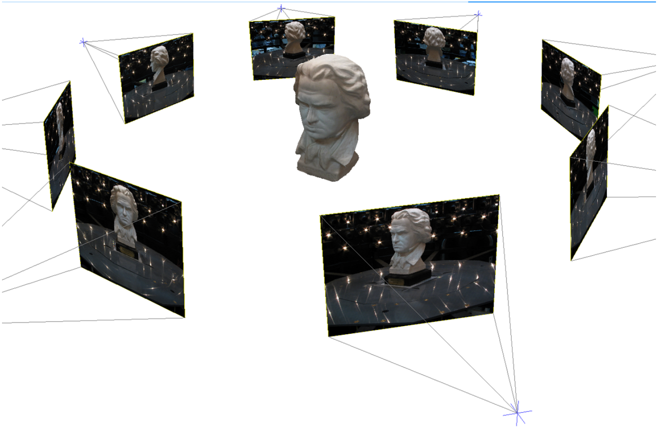
$$E(u) = D(u) + R(u)$$

- **Data term** $D(u)$ measures how well the solution u fits input data.
- **Regularizer** $R(u)$ enforces regularity and smoothness of u.

Minimizing E will give a solution u which fits to the inputs and is smooth!

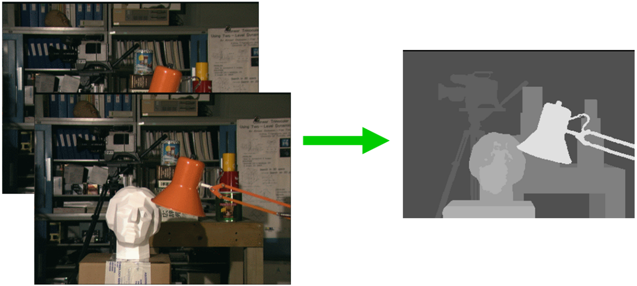
Example: 3D reconstruction

Input: views of an object from different cameras. **Find:** the 3D-object.



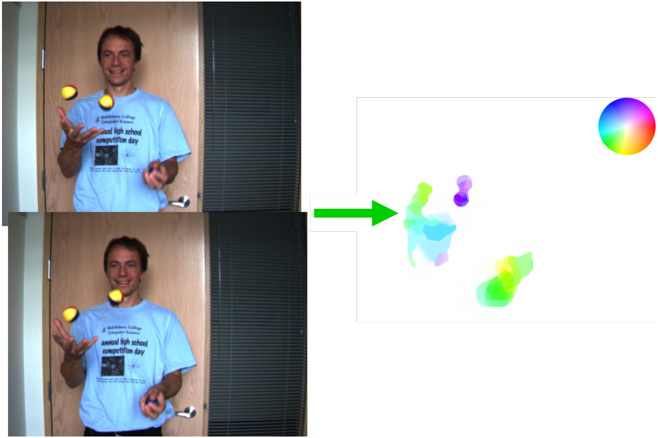
Example: Depth reconstruction

Input: a pair of stereo images. **Find:** the depth in every pixel



Example: Optical flow

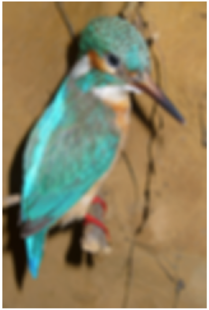
Input: a pair of images. **Find:** displacement vector for each pixel



Example: Image Deblurring

Input: a blurry image. **Find:** a deblurred image.

Original



blurred and noisy

deblurred

Example: Segmentation

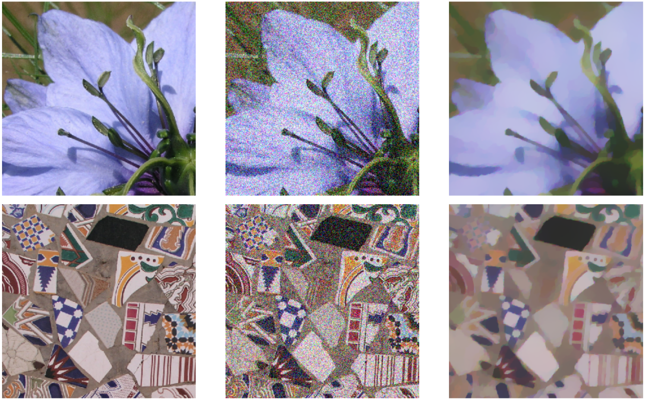
Input: a color image. **Find:** object with certain given characteristics (colors distribution etc.).

Example: Multilabel Segmentation

Input: a color image. **Find:** a meaningful decomposition into several regions.

Image Denoising: The Problem

Input: a noisy image $f : \Omega \to \mathbb{R}^n$. **Find:** denoised $u : \Omega \to \mathbb{R}^n$.



Original *Noisy* *Solution*

Outline

1 Variational methods

2 Image denoising

3 Energy Minimization

Image Denoising: Energy

Data term

- The clean image u must be *similar* to the noisy image f:

$$D(u) := \int_\Omega \big(u(x,y) - f(x,y)\big)^2 \, dx \, dy$$

- Minimize $D(u)$ to guarantee that $u \approx f$.

Regularizer

- Solution u must be noise-free, so we look for *smooth* images u.
- Colors in neighboring pixels must be similar, i.e. $|\nabla u|$ must be small:

$$R(u) := \lambda \int_\Omega \phi\Big(|(\nabla u)(x,y)|\Big) \, dx \, dy.$$

- $\phi : \mathbb{R} \to \mathbb{R}$ is an increasing function, $\lambda > 0$ is a weighting parameter.
- Minimize $R(u)$ to guarantee that $|\nabla u|$ is small, and u noise-free.

Image Denoising: Energy

Data term

- The clean image u must be *similar* to the noisy image f:

$$D(u) := \int_\Omega \big(u(x,y) - f(x,y)\big)^2 \, dx \, dy$$

- Minimize $D(u)$ to guarantee that $u \approx f$.

Regularizer

- Solution u must be noise-free, so we look for *smooth* images u.
- Colors in neighboring pixels must be similar, i.e. $|\nabla u|$ must be small:

$$R(u) := \lambda \int_\Omega \phi\Big(|(\nabla u)(x,y)|\Big) \, dx \, dy.$$

- $\phi : \mathbb{R} \to \mathbb{R}$ is an increasing function, $\lambda > 0$ is a weighting parameter.
- Minimize $R(u)$ to guarantee that $|\nabla u|$ is small, and u noise-free.

Image Denoising: Energy

Data term

- The clean image u must be *similar* to the noisy image f:

$$D(u) := \int_\Omega \big(u(x,y) - f(x,y)\big)^2 \, dx \, dy$$

- Minimize $D(u)$ to guarantee that $u \approx f$.

Regularizer

- Solution u must be noise-free, so we look for *smooth* images u.
- Colors in neighboring pixels must be similar, i.e. $|\nabla u|$ must be small:

$$R(u) := \lambda \int_\Omega \phi\Big(|(\nabla u)(x,y)|\Big) \, dx \, dy.$$

- $\phi : \mathbb{R} \to \mathbb{R}$ is an increasing function, $\lambda > 0$ is a weighting parameter.
- Minimize $R(u)$ to guarantee that $|\nabla u|$ is small, and u noise-free.

Image Denoising: Energy

Data term

- The clean image u must be *similar* to the noisy image f:

$$D(u) := \int_\Omega \big(u(x,y) - f(x,y)\big)^2 \, dx\, dy$$

- Minimize $D(u)$ to guarantee that $u \approx f$.

Regularizer

- Solution u must be noise-free, so we look for *smooth* images u.
- Colors in neighboring pixels must be similar, i.e. $|\nabla u|$ must be small:

$$R(u) := \lambda \int_\Omega \phi\Big(|(\nabla u)(x,y)|\Big) \, dx\, dy.$$

- $\phi : \mathbb{R} \to \mathbb{R}$ is an increasing function, $\lambda > 0$ is a weighting parameter.
- Minimize $R(u)$ to guarantee that $|\nabla u|$ is small, and u noise-free.

Image Denoising: Energy

Data term

- The clean image u must be *similar* to the noisy image f:

$$D(u) := \int_\Omega \big(u(x,y) - f(x,y)\big)^2 \, dx \, dy$$

- Minimize $D(u)$ to guarantee that $u \approx f$.

Regularizer

- Solution u must be noise-free, so we look for *smooth* images u.
- Colors in neighboring pixels must be similar, i.e. $|\nabla u|$ must be small:

$$R(u) := \lambda \int_\Omega \phi\Big(|(\nabla u)(x,y)|\Big) \, dx \, dy.$$

- $\phi : \mathbb{R} \to \mathbb{R}$ is an increasing function, $\lambda > 0$ is a weighting parameter.
- Minimize $R(u)$ to guarantee that $|\nabla u|$ is small, and u noise-free.

Image Denoising: Energy

Data term

- The clean image u must be *similar* to the noisy image f:

$$D(u) := \int_\Omega \big(u(x,y) - f(x,y)\big)^2 \, dx\, dy$$

- Minimize $D(u)$ to guarantee that $u \approx f$.

Regularizer

- Solution u must be noise-free, so we look for *smooth* images u.
- Colors in neighboring pixels must be similar, i.e. $|\nabla u|$ must be small:

$$R(u) := \lambda \int_\Omega \phi\Big(|(\nabla u)(x,y)|\Big) \, dx\, dy.$$

- $\phi : \mathbb{R} \to \mathbb{R}$ is an increasing function, $\lambda > 0$ is a weighting parameter.
- Minimize $R(u)$ to guarantee that $|\nabla u|$ is small, and u noise-free.

Image Denoising: Energy

Denoising energy

$$E(u) = \int_\Omega \Bigg(\underbrace{\big(u(x,y) - f(x,y)\big)^2}_{D(u)} + \underbrace{\lambda\,\phi\Big(\big|(\nabla u)(x,y)\big|\Big)}_{R(u)} \Bigg)\, dx\, dy$$

If $u = f$:
Perfect fit for data: $D(u) = 0$. But u noisy: $R(u) \gg 1$.

If $u = const$:
Bad fit for data: $D(u) \gg 1$. But u smooth: $R(u) = 0$.

True solution
Will be a *trade-off* between data fitting and smoothness.
λ controls the desired degree of smoothness of u.

Image Denoising: Energy

Denoising energy

$$E(u) = \int_\Omega \Big(\underbrace{\big(u(x,y) - f(x,y)\big)^2}_{D(u)} + \underbrace{\lambda\,\phi\Big(\big|(\nabla u)(x,y)\big|\Big)}_{R(u)} \Big)\, dx\, dy$$

If $u = f$:
Perfect fit for data: $D(u) = 0$. But u noisy: $R(u) \gg 1$.

If $u = const$:
Bad fit for data: $D(u) \gg 1$. But u smooth: $R(u) = 0$.

True solution
Will be a *trade-off* between data fitting and smoothness.
λ controls the desired degree of smoothness of u.

Image Denoising: Energy

Denoising energy

$$E(u) = \int_\Omega \Big(\underbrace{\big(u(x,y) - f(x,y)\big)^2}_{D(u)} + \underbrace{\lambda\, \phi\big(|(\nabla u)(x,y)|\big)}_{R(u)} \Big)\, dx\, dy$$

If $u = f$:
Perfect fit for data: $D(u) = 0$. But u noisy: $R(u) \gg 1$.

If $u = const$:
Bad fit for data: $D(u) \gg 1$. But u smooth: $R(u) = 0$.

True solution
Will be a *trade-off* between data fitting and smoothness.
λ controls the desired degree of smoothness of u.

Outline

1. Variational methods
2. Image denoising
3. Energy Minimization

Energy Minimization: Methods

Denoising Energy

$$E(u) = \int_\Omega \left(\big(u(x,y) - f(x,y)\big)^2 + \lambda\,\phi\Big(\big|(\nabla u)(x,y)\big|\Big) \right) dx\,dy$$

How to find the minimizer u in practice?

There are many methods. The most common ones are:

1. Gradient descent: Go along the negative “gradient” of the energy.
2. Euler-Lagrange equation: Necessary condition for the minimizers.
3. Primal-dual methods: Very flexible iterative algorithms.

Energy Minimization: Methods

Denoising Energy

$$E(u) = \int_\Omega \Big(\big(u(x,y) - f(x,y)\big)^2 + \lambda\,\phi\big(|(\nabla u)(x,y)|\big) \Big)\, dx\, dy$$

How to find the minimizer *u* in practice?

There are many methods. The most common ones are:

1. Gradient descent: Go along the negative "gradient" of the energy.
2. Euler-Lagrange equation: Necessary condition for the minimizers.
3. Primal-dual methods: Very flexible iterative algorithms.

Energy Minimization: Methods

Denoising Energy

$$E(u) = \int_\Omega \left(\big(u(x,y) - f(x,y)\big)^2 + \lambda \phi\Big(\big|(\nabla u)(x,y)\big|\Big) \right) dx\, dy$$

How to find the minimizer u in practice?

There are many methods. The most common ones are:

1. Gradient descent: Go along the negative “gradient” of the energy.
2. Euler-Lagrange equation: Necessary condition for the minimizers.
3. Primal-dual methods: Very flexible iterative algorithms.

Energy Minimization: Methods

Denoising Energy

$$E(u) = \int_\Omega \left(\big(u(x,y) - f(x,y)\big)^2 + \lambda\,\phi\Big(\big|(\nabla u)(x,y)\big|\Big) \right) dx\,dy$$

How to find the minimizer u in practice?

There are many methods. The most common ones are:

1. Gradient descent: Go along the negative “gradient” of the energy.
2. Euler-Lagrange equation: Necessary condition for the minimizers.
3. Primal-dual methods: Very flexible iterative algorithms.

Gradient Descent: Gradient of the Energy

Intuitively: $(\nabla E)(u)$ is the gradient w.r.t. values $u(x, y)$ at each (x, y).

Analogy with finite $e : \mathbb{R}^k \to \mathbb{R}$:

- For $z \in \mathbb{R}^k$: $(\nabla e)(z)$ has $(\dim \mathbb{R}^k)$-many components.
- If the position z is changed slightly to $z + h$, then $(\nabla e)(z)$ describes the rate of the change of e:

$$e(z + h) \approx e(z) + \sum_{i=1}^{k} \big((\nabla e)(z)\big)_i \cdot h_i$$

Therefore:

- For $u : \Omega \to \mathbb{R}$: $(\nabla E)(u)$ has $(\dim \{\hat{u} : \Omega \to \mathbb{R}\})$-many components. So $(\nabla E)(u)$ **is a function** $(\nabla E)(u) : \Omega \to \mathbb{R}$.
- If the image u is changed slightly in each pixel to $u(x, y) + h(x, y)$, then $(\nabla E)(u)$ describes the rate of the change of E:

$$E(u + h) \approx E(u) + \int_\Omega \big((\nabla E)(u)\big)(x, y) \cdot h(x, y)\, dx\, dy$$

Gradient Descent: Gradient of the Energy

Intuitively: $(\nabla E)(u)$ is the gradient w.r.t. values $u(x, y)$ at each (x, y).
Analogy with finite $e : \mathbb{R}^k \to \mathbb{R}$:

- For $z \in \mathbb{R}^k$: $(\nabla e)(z)$ has $(\dim \mathbb{R}^k)$-many components.
- If the position z is changed slightly to $z + h$,
  then $(\nabla e)(z)$ describes the rate of the change of e:

$$e(z + h) \approx e(z) + \sum_{i=1}^{k} \big((\nabla e)(z)\big)_i \cdot h_i$$

Therefore:

- For $u : \Omega \to \mathbb{R}$: $(\nabla E)(u)$ has $(\dim \{\hat{u} : \Omega \to \mathbb{R}\})$-many components.
  So $(\nabla E)(u)$ **is a function** $(\nabla E)(u) : \Omega \to \mathbb{R}$.
- If the image u is changed slightly in each pixel to $u(x, y) + h(x, y)$, then
  $(\nabla E)(u)$ describes the rate of the change of E:

$$E(u + h) \approx E(u) + \int_\Omega \big((\nabla E)(u)\big)(x, y) \cdot h(x, y)\, dx\, dy$$

Gradient Descent: Gradient of the Energy

Intuitively: $(\nabla E)(u)$ is the gradient w.r.t. values $u(x,y)$ at each (x,y).
Analogy with finite $e : \mathbb{R}^k \to \mathbb{R}$:

- For $z \in \mathbb{R}^k$: $(\nabla e)(z)$ has $(\dim \mathbb{R}^k)$-many components.
- If the position z is changed slightly to $z+h$, then $(\nabla e)(z)$ describes the rate of the change of e:

$$e(z+h) \approx e(z) + \sum_{i=1}^{k} \big((\nabla e)(z)\big)_i \cdot h_i$$

Therefore:

- For $u : \Omega \to \mathbb{R}$: $(\nabla E)(u)$ has $(\dim \{\hat{u} : \Omega \to \mathbb{R}\})$-many components. So $(\nabla E)(u)$ **is a function** $(\nabla E)(u) : \Omega \to \mathbb{R}$.
- If the image u is changed slightly in each pixel to $u(x,y) + h(x,y)$, then $(\nabla E)(u)$ describes the rate of the change of E:

$$E(u+h) \approx E(u) + \int_\Omega \big((\nabla E)(u)\big)(x,y) \cdot h(x,y)\, dx\, dy$$

Gradient Descent: Gradient of the Energy

Intuitively: $(\nabla E)(u)$ is the gradient w.r.t. values $u(x, y)$ at each (x, y).
Analogy with finite $e : \mathbb{R}^k \to \mathbb{R}$**:**

- For $z \in \mathbb{R}^k$: $(\nabla e)(z)$ has $(\dim \mathbb{R}^k)$-many components.
- If the position z is changed slightly to $z + h$, then $(\nabla e)(z)$ describes the rate of the change of e:

$$e(z + h) \approx e(z) + \sum_{i=1}^{k} \big((\nabla e)(z)\big)_i \cdot h_i$$

Therefore:

- For $u : \Omega \to \mathbb{R}$: $(\nabla E)(u)$ has $(\dim \{\hat{u} : \Omega \to \mathbb{R}\})$-many components. So $(\nabla E)(u)$ **is a function** $(\nabla E)(u) : \Omega \to \mathbb{R}$.
- If the image u is changed slightly in each pixel to $u(x, y) + h(x, y)$, then $(\nabla E)(u)$ describes the rate of the change of E:

$$E(u + h) \approx E(u) + \int_\Omega \big((\nabla E)(u)\big)(x, y) \cdot h(x, y)\, dx\, dy$$

Gradient Descent: Gradient of the Energy

Intuitively: $(\nabla E)(u)$ is the gradient w.r.t. values $u(x,y)$ at each (x,y).
Analogy with finite $e : \mathbb{R}^k \to \mathbb{R}$:

- For $z \in \mathbb{R}^k$: $(\nabla e)(z)$ has $(\dim \mathbb{R}^k)$-many components.
- If the position z is changed slightly to $z+h$, then $(\nabla e)(z)$ describes the rate of the change of e:

$$e(z+h) \approx e(z) + \sum_{i=1}^{k} \big((\nabla e)(z)\big)_i \cdot h_i$$

Therefore:

- For $u : \Omega \to \mathbb{R}$: $(\nabla E)(u)$ has $(\dim \{\hat{u} : \Omega \to \mathbb{R}\})$-many components. So $(\nabla E)(u)$ **is a function** $(\nabla E)(u) : \Omega \to \mathbb{R}$.
- If the image u is changed slightly in each pixel to $u(x,y) + h(x,y)$, then $(\nabla E)(u)$ describes the rate of the change of E:

$$E(u+h) \approx E(u) + \int_\Omega \big((\nabla E)(u)\big)(x,y) \cdot h(x,y)\, dx\, dy$$

Gradient Descent: Update Equation

Idea

- The gradient is the direction of steepest increase of E.
- The *negative* gradient is the direction is *steepest descent*.

Gradient descent equation

$$\partial_t u = -(\nabla E)(u)$$

So, having computed some candidate u with energy $E(u)$, we can construct a better candidate u_{new} with a *potentially lower* energy $E(u_{\text{new}})$:

$$(u_{\text{new}})(x, y) = u(x, y) + \tau \left(-(\nabla E(u))(x, y) \right)$$

Gradient Descent: Update Equation

Idea

- The gradient is the direction of steepest increase of E.
- The *negative* gradient is the direction is *steepest descent*.

Gradient descent equation

$$\partial_t u = -(\nabla E)(u)$$

So, having computed some candidate u with energy $E(u)$, we can construct a better candidate u_{new} with a *potentially lower* energy $E(u_{\text{new}})$:

$$(u_{\text{new}})(x, y) = u(x, y) + \tau \left(- (\nabla E(u))(x, y) \right)$$

Gradient Descent: Update Equation

Idea

- The gradient is the direction of steepest increase of E.
- The *negative* gradient is the direction is *steepest descent*.

Gradient descent equation

$$\partial_t u = -(\nabla E)(u)$$

So, having computed some candidate u with energy $E(u)$, we can construct a better candidate u_{new} with a *potentially lower* energy $E(u_{\text{new}})$:

$$(u_{\text{new}})(x, y) = u(x, y) + \tau \Big(-(\nabla E(u))(x, y) \Big)$$

Gradient Descent: Update Equation

Idea

- The gradient is the direction of steepest increase of E.
- The *negative* gradient is the direction is *steepest descent*.

Gradient descent equation

$$\partial_t u = -(\nabla E)(u)$$

So, having computed some candidate u with energy $E(u)$, we can construct a better candidate u_{new} with a *potentially lower* energy $E(u_{\text{new}})$:

$$(u_{\text{new}})(x, y) = u(x, y) + \tau \Big(- (\nabla E(u))(x, y) \Big)$$

Gradient Descent: Image Denoising

Denoising energy

$$E(u) = \int_\Omega \left(\big(u(x,y) - f(x,y)\big)^2 + \lambda\,\phi\Big(\big|(\nabla u)(x,y)\big|\Big) \right) dx\,dy$$

Functional derivative

$$(\nabla E)(u) = 2(u - f) - \lambda\,\mathrm{div}\left(\frac{\phi'(|\nabla u|)}{|\nabla u|}\nabla u\right)$$

Gradient descent equation

$$\partial_t u = -(\nabla E)(u) = 2(f - u) + \lambda\,\mathrm{div}\left(\frac{\phi'(|\nabla u|)}{|\nabla u|}\nabla u\right)$$

Observe:

- The structure of the equation is the same as for *diffusion* with diffusivity $g := \lambda\,\frac{\phi'(|\nabla u|)}{|\nabla u|}$, but with an additional term $2(f - u)$.

Gradient Descent: Image Denoising

Denoising energy

$$E(u) = \int_\Omega \left(\left(u(x,y) - f(x,y)\right)^2 + \lambda \, \phi\left(|(\nabla u)(x,y)| \right) \right) dx\, dy$$

Functional derivative

$$(\nabla E)(u) = 2(u - f) - \lambda \operatorname{div}\left(\frac{\phi'\left(|\nabla u|\right)}{|\nabla u|} \nabla u \right)$$

Gradient descent equation

$$\partial_t u = -(\nabla E)(u) = 2(f - u) + \lambda \operatorname{div}\left(\frac{\phi'\left(|\nabla u|\right)}{|\nabla u|} \nabla u \right)$$

Observe:

- The structure of the equation is the same as for *diffusion* with diffusivity $g := \lambda \frac{\phi'(|\nabla u|)}{|\nabla u|}$, but with an additional term $2(f - u)$.

Gradient Descent: Image Denoising

Denoising energy

$$E(u) = \int_\Omega \Big(\big(u(x,y) - f(x,y)\big)^2 + \lambda\, \phi\big(|(\nabla u)(x,y)|\big) \Big)\, dx\, dy$$

Functional derivative

$$(\nabla E)(u) = 2(u - f) - \lambda\, \mathrm{div}\left(\frac{\phi'(|\nabla u|)}{|\nabla u|} \nabla u \right)$$

Gradient descent equation

$$\partial_t u = -(\nabla E)(u) = 2(f - u) + \lambda\, \mathrm{div}\left(\frac{\phi'(|\nabla u|)}{|\nabla u|} \nabla u \right)$$

Observe:

- The structure of the equation is the same as for *diffusion* with diffusivity $g := \lambda \frac{\phi'(|\nabla u|)}{|\nabla u|}$, but with an additional term $2(f - u)$.

Gradient Descent: Image Denoising

Denoising energy

$$E(u) = \int_\Omega \left(\big(u(x,y) - f(x,y)\big)^2 \; + \; \lambda \, \phi\Big(\big|(\nabla u)(x,y)\big| \Big) \right) dx\, dy$$

Functional derivative

$$(\nabla E)(u) = 2(u - f) - \lambda \operatorname{div} \left(\frac{\phi'\big(|\nabla u|\big)}{|\nabla u|} \nabla u \right)$$

Gradient descent equation

$$\partial_t u = -(\nabla E)(u) = 2(f - u) + \lambda \operatorname{div} \left(\frac{\phi'\big(|\nabla u|\big)}{|\nabla u|} \nabla u \right)$$

Observe:

- The structure of the equation is the same as for *diffusion* with diffusivity $g := \lambda \frac{\phi'(|\nabla u|)}{|\nabla u|}$, but with an additional term $2(f - u)$.

Gradient Descent: Quadratic Regularizer Example

Quadratic regularizer: Set $\phi(s) := \frac{1}{2}s^2$.

Denoising energy

$$E(u) = \int_\Omega \left(\left(u(x,y) - f(x,y)\right)^2 + \frac{\lambda}{2} |(\nabla u)(x,y)|^2 \right) dx\, dy$$

Using this regularizer leads to oversmoothing, solutions are too blurry.

Gradient descent equation

We have $\frac{\phi'(s)}{s} = 1$, therefore

$$\partial_t u = 2(f - u) + \lambda \Delta u$$

Gradient Descent: Quadratic Regularizer Example

Quadratic regularizer: Set $\phi(s) := \frac{1}{2}s^2$.

Denoising energy

$$E(u) = \int_\Omega \left(\big(u(x,y) - f(x,y)\big)^2 + \frac{\lambda}{2} |(\nabla u)(x,y)|^2 \right) dx\, dy$$

Using this regularizer leads to oversmoothing, solutions are too blurry.

Gradient descent equation

We have $\frac{\phi'(s)}{s} = 1$, therefore

$$\partial_t u = 2(f - u) + \lambda \Delta u$$

Gradient Descent: Quadratic Regularizer Example

Quadratic regularizer: Set $\phi(s) := \frac{1}{2}s^2$.

Denoising energy

$$E(u) = \int_\Omega \left(\big(u(x,y) - f(x,y)\big)^2 + \frac{\lambda}{2} |(\nabla u)(x,y)|^2 \right) dx\, dy$$

Using this regularizer leads to oversmoothing, solutions are too blurry.

Gradient descent equation

We have $\frac{\phi'(s)}{s} = 1$, therefore

$$\partial_t u = 2(f - u) + \lambda \Delta u$$

Gradient Descent: Huber Regularizer Example

Huber regularizer: Set $\phi(s) := h_\varepsilon(s) := \begin{Bmatrix} \frac{s^2}{2\varepsilon} & \text{if } s < \varepsilon \\ s - \frac{\varepsilon}{2} & \text{else} \end{Bmatrix}.$

Denoising energy

$$E(u) = \int_\Omega \left(\left(u(x,y) - f(x,y)\right)^2 + \lambda h_\varepsilon\left(|(\nabla u)(x,y)|\right) \right) dx\, dy$$

This regularizer only smooths in flat regions, edges are well preserved.

Gradient descent equation

We have $\frac{\phi'(s)}{s} = \frac{1}{\max(\varepsilon, s)}$, therefore

$$\partial_t u = 2(f - u) + \lambda \operatorname{div}\left(\frac{1}{\max(\varepsilon, |\nabla u|)} \nabla u \right)$$

Gradient Descent: Huber Regularizer Example

Huber regularizer: Set $\phi(s) := h_\varepsilon(s) := \begin{Bmatrix} \frac{s^2}{2\varepsilon} & \text{if } s < \varepsilon \\ s - \frac{\varepsilon}{2} & \text{else} \end{Bmatrix}$.

Denoising energy

$$E(u) = \int_\Omega \left((u(x,y) - f(x,y))^2 + \lambda h_\varepsilon\left(|(\nabla u)(x,y)|\right) \right) dx\, dy$$

This regularizer only smooths in flat regions, edges are well preserved.

Gradient descent equation

We have $\frac{\phi'(s)}{s} = \frac{1}{\max(\varepsilon, s)}$, therefore

$$\partial_t u = 2(f - u) + \lambda \operatorname{div}\left(\frac{1}{\max(\varepsilon, |\nabla u|)} \nabla u \right)$$

Gradient Descent: Huber Regularizer Example

Huber regularizer: Set $\phi(s) := h_\varepsilon(s) := \begin{Bmatrix} \frac{s^2}{2\varepsilon} & \text{if } s < \varepsilon \\ s - \frac{\varepsilon}{2} & \text{else} \end{Bmatrix}.$

Denoising energy

$$E(u) = \int_\Omega \Big(\big(u(x,y) - f(x,y)\big)^2 + \lambda h_\varepsilon\big(|(\nabla u)(x,y)|\big) \Big)\, dx\, dy$$

This regularizer only smooths in flat regions, edges are well preserved.

Gradient descent equation

We have $\frac{\phi'(s)}{s} = \frac{1}{\max(\varepsilon, s)}$, therefore

$$\partial_t u = 2(f-u) + \lambda \operatorname{div}\left(\frac{1}{\max(\varepsilon, |\nabla u|)} \nabla u \right)$$

Euler-Lagrange Equation

Idea

Setting the gradient to zero, i.e. considering $(\nabla E)(u) = 0$, yields a *necessary optimality condition* for the minimizers u.

Euler-Lagrange equation

$$2(u - f) - \lambda \operatorname{div}\left(\frac{\phi'(|\nabla u|)}{|\nabla u|}\nabla u\right) = 0$$

For convex energies:

Any image u fulfilling the equation is a minimizer of the energy.

Solving:

- discretize
- apply fixed-point iteration

Euler-Lagrange Equation

Idea
Setting the gradient to zero, i.e. considering $(\nabla E)(u) = 0$, yields a *necessary optimality condition* for the minimizers u.

Euler-Lagrange equation

$$2(u - f) - \lambda \operatorname{div}\left(\frac{\phi'\left(|\nabla u|\right)}{|\nabla u|}\nabla u\right) = 0$$

For convex energies:
Any image u fulfilling the equation is a minimizer of the energy.

Solving:

- discretize
- apply fixed-point iteration

Euler-Lagrange Equation

Idea
Setting the gradient to zero, i.e. considering $(\nabla E)(u) = 0$, yields a *necessary optimality condition* for the minimizers *u*.

Euler-Lagrange equation

$$2(u - f) - \lambda \operatorname{div}\left(\frac{\phi'\left(|\nabla u|\right)}{|\nabla u|}\nabla u\right) = 0$$

For convex energies:
Any image *u* fulfilling the equation is a minimizer of the energy.

Solving:

- discretize
- apply fixed-point iteration

Euler-Lagrange Equation

Idea

Setting the gradient to zero, i.e. considering $(\nabla E)(u) = 0$, yields a *necessary optimality condition* for the minimizers u.

Euler-Lagrange equation

$$2(u - f) - \lambda \operatorname{div}\left(\frac{\phi'(|\nabla u|)}{|\nabla u|}\nabla u\right) = 0$$

For convex energies:

Any image u fulfilling the equation is a minimizer of the energy.

Solving:

- discretize
- apply fixed-point iteration

Euler-Lagrange Equation

Idea

Setting the gradient to zero, i.e. considering $(\nabla E)(u) = 0$, yields a *necessary optimality condition* for the minimizers u.

Euler-Lagrange equation

$$2(u - f) - \lambda \operatorname{div} \left(\frac{\phi'\left(|\nabla u|\right)}{|\nabla u|} \nabla u \right) = 0$$

For convex energies:

Any image u fulfilling the equation is a minimizer of the energy.

Solving:

- discretize
- apply fixed-point iteration

Euler-Lagrange Equation: Discretization

Forward differences for the diffusivity $g := \widehat{g}\big(|\nabla^+ u|\big)$, $\widehat{g}(s) := \frac{\phi'(s)}{s}$.
Forward differences for ∇, backward differences for div:

$$2(u - f) - \lambda \, \mathrm{div}^- \left(g \, \nabla^+ u\right) = 0.$$

Fully written out, this is

$$2(u-f) - \lambda \Big(g_r \, u(x+1,y) + g_l \, u(x-1,y) + g_u \, u(x,y+1) + g_d \, u(x,y-1) - (g_r + g_l + g_u + g_d) \, u(x,y) \Big) = 0$$

with

$$g_r := \mathbb{1}_{x+1<W} \cdot g(x,y), \qquad g_l := \mathbb{1}_{x>0} \cdot g(x-1,y),$$
$$g_u := \mathbb{1}_{y+1<H} \cdot g(x,y), \qquad g_d := \mathbb{1}_{y>0} \cdot g(x,y-1).$$

This is a nonlinear equations system. Use a fixed point iteration scheme.

Euler-Lagrange Equation: Discretization

Forward differences for the diffusivity $g := \widehat{g}\left(|\nabla^+ u|\right)$, $\widehat{g}(s) := \frac{\phi'(s)}{s}$.
Forward differences for ∇, backward differences for div:

$$2(u - f) - \lambda \operatorname{div}^- \left(g \nabla^+ u\right) = 0.$$

Fully written out, this is

$$\begin{aligned} 2(u - f) - \lambda \Big(\quad & g_r\, u(x+1, y) + g_l\, u(x-1, y) \\ & + g_u\, u(x, y+1) + g_d\, u(x, y-1) \\ & - (g_r + g_l + g_u + g_d)\, u(x, y) \quad \Big) = 0 \end{aligned}$$

with

$$g_r := \mathbf{1}_{x+1<W} \cdot g(x, y), \qquad g_l := \mathbf{1}_{x>0} \cdot g(x-1, y),$$
$$g_u := \mathbf{1}_{y+1<H} \cdot g(x, y), \qquad g_d := \mathbf{1}_{y>0} \cdot g(x, y-1).$$

This is a nonlinear equations system. Use a fixed point iteration scheme.

Euler-Lagrange Equation: Discretization

Forward differences for the diffusivity $g := \widehat{g}\big(|\nabla^+ u|\big)$, $\widehat{g}(s) := \frac{\phi'(s)}{s}$.
Forward differences for ∇, backward differences for div:

$$2(u - f) - \lambda \operatorname{div}^- \big(g \nabla^+ u\big) = 0.$$

Fully written out, this is

$$2(u - f) - \lambda\Big(\quad g_r\, u(x+1, y) + g_l\, u(x-1, y) + g_u\, u(x, y+1) + g_d\, u(x, y-1) - (g_r + g_l + g_u + g_d)\, u(x, y) \quad \Big) = 0$$

with

$$g_r := \mathbf{1}_{x+1<W} \cdot g(x, y), \qquad g_l := \mathbf{1}_{x>0} \cdot g(x-1, y),$$
$$g_u := \mathbf{1}_{y+1<H} \cdot g(x, y), \qquad g_d := \mathbf{1}_{y>0} \cdot g(x, y-1).$$

This is a nonlinear equations system. Use a fixed point iteration scheme.

Euler-Lagrange Equation: Fixed-Point Iteration

1. Start with an image u^0.

2. Compute the diffusivity $g = \widehat{g}(|\nabla^+ u^k|)$ at the current iterate u^k.
   Compute g_r, g_l, g_u, g_d in each pixel (see previous slide).

3. Solve the following *linear* system for u^{k+1}: for all $(x, y) \in \Omega$,

$$\Big(2 + \lambda(g_r + g_l + g_u + g_d)\Big)\, u^{k+1}(x, y)$$
$$- \lambda\, g_r\, u^{k+1}(x + 1, y) - \lambda\, g_l\, u^{k+1}(x - 1, y)$$
$$- \lambda\, g_u\, u^{k+1}(x, y + 1) - \lambda\, g_d\, u^{k+1}(x, y - 1) = 2f(x, y).$$

4. Iterate until convergence.

Euler-Lagrange Equation: Fixed-Point Iteration

1. Start with an image u^0.

2. Compute the diffusivity $g = \widehat{g}\left(|\nabla^+ u^k|\right)$ at the current iterate u^k.
   Compute g_r, g_l, g_u, g_d in each pixel (see previous slide).

3. Solve the following *linear* system for u^{k+1}: for all $(x, y) \in \Omega$,

$$\begin{aligned}&\Big(2 + \lambda(g_r + g_l + g_u + g_d)\Big)\, u^{k+1}(x, y)\\ &\quad - \lambda\, g_r\, u^{k+1}(x+1, y) - \lambda\, g_l\, u^{k+1}(x-1, y)\\ &\quad - \lambda\, g_u\, u^{k+1}(x, y+1) - \lambda\, g_d\, u^{k+1}(x, y-1) = 2f(x, y).\end{aligned}$$

4. Iterate until convergence.

Euler-Lagrange Equation: Fixed-Point Iteration

1. Start with an image u^0.

2. Compute the diffusivity $g = \widehat{g}\left(|\nabla^+ u^k|\right)$ at the current iterate u^k.
   Compute g_r, g_l, g_u, g_d in each pixel (see previous slide).

3. Solve the following *linear* system for u^{k+1}: for all $(x, y) \in \Omega$,

$$\begin{aligned}&\Big(2 + \lambda(g_r + g_l + g_u + g_d)\Big)\, u^{k+1}(x, y)\\&\quad - \lambda\, g_r\, u^{k+1}(x+1, y) - \lambda\, g_l\, u^{k+1}(x-1, y)\\&\quad - \lambda\, g_u\, u^{k+1}(x, y+1) - \lambda\, g_d\, u^{k+1}(x, y-1) = 2f(x, y).\end{aligned}$$

4. Iterate until convergence.

Euler-Lagrange Equation: Fixed-Point Iteration

1. Start with an image u^0.

2. Compute the diffusivity $g = \widehat{g}\big(|\nabla^+ u^k|\big)$ at the current iterate u^k.
   Compute g_r, g_l, g_u, g_d in each pixel (see previous slide).

3. Solve the following *linear* system for u^{k+1}: for all $(x, y) \in \Omega$,

$$\begin{aligned}&\Big(2 + \lambda(g_r + g_l + g_u + g_d)\Big)\, u^{k+1}(x, y)\\ &\quad - \lambda\, g_r\, u^{k+1}(x+1, y) - \lambda\, g_l\, u^{k+1}(x-1, y)\\ &\quad - \lambda\, g_u\, u^{k+1}(x, y+1) - \lambda\, g_d\, u^{k+1}(x, y-1) = 2f(x, y).\end{aligned}$$

4. Iterate until convergence.

Linear Equation Systems: Jacobi Method

Jacobi Method

To solve $Az = b$: split $A = D + R$ with diagonal D and off-diagonal R:

$$D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & a_{nn} \end{pmatrix}, \quad R = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ a_{n1} & \dots & a_{n,n-1} & 0 \end{pmatrix}$$

$(D + R)z = b$, so $z = D^{-1}(b - Rz)$. One iteration leads to the update:

$$z_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} z_j^k \right)$$

Update for the Euler-Lagrange equation

$$u^{k+1}(x, y) = \frac{2f(x,y) + \lambda\, g_r u^k(x+1,y) + \lambda\, g_l u^k(x-1,y) + \lambda\, g_u u^k(x,y+1) + \lambda\, g_d u^k(x,y-1)}{2 + \lambda\,(g_r + g_l + g_u + g_d)}$$

Linear Equation Systems: Jacobi Method

Jacobi Method

To solve $Az = b$: split $A = D + R$ with diagonal D and off-diagonal R:

$$D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & a_{nn} \end{pmatrix}, \quad R = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ a_{n1} & \dots & a_{n,n-1} & 0 \end{pmatrix}$$

$(D + R)z = b$, so $z = D^{-1}(b - Rz)$. One iteration leads to the update:

$$z_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} z_j^k \right)$$

Update for the Euler-Lagrange equation

$$u^{k+1}(x, y) = \frac{2f(x,y) + \lambda\, g_r u^k(x+1,y) + \lambda\, g_l u^k(x-1,y) + \lambda\, g_u u^k(x,y+1) + \lambda\, g_d u^k(x,y-1)}{2 + \lambda\,(g_r + g_l + g_u + g_d)}$$

Linear Equation Systems: Jacobi Method

Jacobi Method
To solve $Az = b$: split $A = D + R$ with diagonal D and off-diagonal R:

$$D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & a_{nn} \end{pmatrix}, \quad R = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ a_{n1} & \dots & a_{n,n-1} & 0 \end{pmatrix}$$

$(D + R)z = b$, so $z = D^{-1}(b - Rz)$. One iteration leads to the update:

$$z_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} z_j^k \right)$$

Update for the Euler-Lagrange equation

$$u^{k+1}(x, y) = \frac{2f(x,y) + \lambda\, g_r u^k(x+1,y) + \lambda\, g_l u^k(x-1,y) + \lambda\, g_u u^k(x,y+1) + \lambda\, g_d u^k(x,y-1)}{2 + \lambda\, (g_r + g_l + g_u + g_d)}$$

Linear Equation Systems: Gauss-Seidel Method

Gauss-Seidel Method

Split $A = L_* + U$, with L_* lower triangular and U upper triangular:

$$L_* = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ a_{n1} & \dots & a_{n,n-1} & a_{nn} \end{pmatrix}, \quad U = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ 0 & \dots & 0 & 0 \end{pmatrix}$$

$(L_* + U)z = b$, so $z = L_*^{-1}(b - Ux)$. One iteration leads to the update:

$$z_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j>i} a_{ij} z_j^k - \sum_{j<i} a_{ij} z_j^{k+1}\right)$$

This is *exactly* the Jacobi update, but with *new values* z^{k+1} if available.

Red-black scheme

To parallelize the Gauss-Seidel update: *First:* update only at pixels (x, y) with $(x + y)\%2 = 0$. *Then:* only with $(x + y)\%2 = 1$.

Linear Equation Systems: Gauss-Seidel Method

Gauss-Seidel Method

Split $A = L_* + U$, with L_* lower triangular and U upper triangular:

$$L_* = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ a_{n1} & \dots & a_{n,n-1} & a_{nn} \end{pmatrix}, \; U = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ 0 & \dots & 0 & 0 \end{pmatrix}$$

$(L_* + U)z = b$, so $z = L_*^{-1}(b - Ux)$. One iteration leads to the update:

$$z_i^{k+1} = \frac{1}{a_{ii}} \Big(b_i - \sum_{j>i} a_{ij} z_j^k - \sum_{j<i} a_{ij} z_j^{k+1} \Big)$$

This is *exactly* the Jacobi update, but with *new values* z^{k+1} if available.

Red-black scheme

To parallelize the Gauss-Seidel update: *First:* update only at pixels (x, y) with $(x + y)\%2 = 0$. *Then:* only with $(x + y)\%2 = 1$.

Linear Equation Systems: Gauss-Seidel Method

Gauss-Seidel Method
Split $A = L_* + U$, with L_* lower triangular and U upper triangular:

$$L_* = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ a_{n1} & \dots & a_{n,n-1} & a_{nn} \end{pmatrix}, \; U = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ 0 & \dots & 0 & 0 \end{pmatrix}$$

$(L_* + U)z = b$, so $z = L_*^{-1}(b - Ux)$. One iteration leads to the update:

$$z_i^{k+1} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j>i} a_{ij} z_j^k - \sum_{j<i} a_{ij} z_j^{k+1}\Big)$$

This is *exactly* the Jacobi update, but with *new values* z^{k+1} if available.

Red-black scheme
To parallelize the Gauss-Seidel update: *First:* update only at pixels (x, y) with $(x + y)\%2 = 0$. *Then:* only with $(x + y)\%2 = 1$.

Linear Equation Systems: Gauss-Seidel Method

Gauss-Seidel Method

Split $A = L_* + U$, with L_* lower triangular and U upper triangular:

$$L_* = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ a_{n1} & \dots & a_{n,n-1} & a_{nn} \end{pmatrix}, \; U = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & & \vdots \\ \vdots & & \ddots & a_{n-1,n} \\ 0 & \dots & 0 & 0 \end{pmatrix}$$

$(L_* + U)z = b$, so $z = L_*^{-1}(b - Ux)$. One iteration leads to the update:

$$z_i^{k+1} = \frac{1}{a_{ii}} \Big(b_i - \sum_{j>i} a_{ij} z_j^k - \sum_{j<i} a_{ij} z_j^{k+1} \Big)$$

This is *exactly* the Jacobi update, but with *new values* z^{k+1} if available.

Red-black scheme

To parallelize the Gauss-Seidel update: *First:* update only at pixels (x, y) with $(x + y)\%2 = 0$. *Then:* only with $(x + y)\%2 = 1$.

Linear Equation Systems: Gauss-Seidel Method with SOR

Successive Over-Relaxation (SOR)
Accelerates the Gauss-Seidel method by linear extrapolation.

SOR update step
Let $\bar{z}^{k+1}$ be the result of one Gauss-Seidel iteration applied to z^k.
Compute

$$z^{k+1} = \bar{z}^{k+1} + \theta(\bar{z}^{k+1} - z^k)$$

where $\theta \in [0, 1)$ is a fixed parameter.

Convergence
SOR converges for any $\theta \in [0, 1)$. The optimal θ depends on A.
In practice, one uses values near 1, typically 0.5–0.9, or 0.9–0.98.

Linear Equation Systems: Gauss-Seidel Method with SOR

Successive Over-Relaxation (SOR)
Accelerates the Gauss-Seidel method by linear extrapolation.

SOR update step
Let $\bar{z}^{k+1}$ be the result of one Gauss-Seidel iteration applied to z^k.
Compute

$$z^{k+1} = \bar{z}^{k+1} + \theta(\bar{z}^{k+1} - z^k)$$

where $\theta \in [0, 1)$ is a fixed parameter.

Convergence
SOR converges for any $\theta \in [0, 1)$. The optimal θ depends on A.
In practice, one uses values near 1, typically 0.5–0.9, or 0.9–0.98.

Linear Equation Systems: Gauss-Seidel Method with SOR

Successive Over-Relaxation (SOR)
Accelerates the Gauss-Seidel method by linear extrapolation.

SOR update step
Let $\bar{z}^{k+1}$ be the result of one Gauss-Seidel iteration applied to z^k.
Compute

$$z^{k+1} = \bar{z}^{k+1} + \theta(\bar{z}^{k+1} - z^k)$$

where $\theta \in [0, 1)$ is a fixed parameter.

Convergence
SOR converges for any $\theta \in [0, 1)$. The optimal θ depends on A.
In practice, one uses values near 1, typically 0.5–0.9, or 0.9–0.98.