Image Processing

Continuous Energy Minimization

Idea

Instead to say what should be done (algorithms), it is formulated what for properties the result should have (model).

Realization of an object is represented by mappings (e.g. $\mathbb{R}^2 \rightarrow \mathbb{R}$)

The desired properties of the model are represented by the Energy − a "function" that penalizes inappropriate mappings.

The problem becomes an optimization task – search for the solution of the optimal energy.

Cases:
Domain of definition: continuous, discrete
Range: continuous, discrete

Today: continuous range, both discrete and continuous domain

Discrete domain of definition

Example – denoising

$R \in \mathbb{Z}^2$ – the set of pixels, $E \subset R^2$ – the neighborhood structure
$x : R \to \mathbb{Z}$ – the initial image (gray-valued for simplicity)
$y : R \to \mathbb{R}$ – the unknown (e.g. the restored image)

The energy (usually) consists of two terms:

- The data term:

$$E_d(y) = \sum_{r \in R} (x_r - y_r)^2$$

  (the solution should be as similar as possible to the original)

- The model term:

$$E_m(y) = \sum_{rr' \in E} (y_r - y_{r'})^2$$

  (the solution should be smooth)

Discrete domain of definition

The optimization task is:

$$y^* = \arg\min_y \left[E_d(y) + \alpha E_m(y) \right]$$

(search for an agreement)

Solution – derive, set to zero, resolve …

For a particular pixel r^*: consider parts (addends) of the energy that depend on r^*

$$\frac{\partial}{\partial y_{r^*}} \left[\sum_{r \in R} (x_r - y_r)^2 + \alpha \sum_{rr' \in E} (y_r - y_{r'})^2 \right] =$$

$$y_{r^*} - x_{r^*} + \alpha \sum_{r' : r^* r' \in E} (y_{r^*} - y'_r) = 0$$

It follows:

$$(1 + 4\alpha) y_{ij} - \alpha y_{ij-1} - \alpha y_{ij+1} - \alpha y_{i-1j} - \alpha y_{i+1j} = x_{ij} \quad \forall i, j$$

Discrete domain of definition

$$(1+4\alpha)y_{ij} - \alpha y_{ij-1} - \alpha y_{ij+1} - \alpha y_{i-1j} - \alpha y_{i+1j} = x_{ij} \quad \forall i,j$$

The system of linear equations with $n = |R|$ variables and n equations

$$A \cdot y = x$$

with

$y = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$ – the solution

$x = (x_1, x_2, \ldots, x_n) \in \mathbb{Z}^n$ – the original image

Elements of the matrix are:

$a_{ii} = 1 + 4\alpha$ and $a_{ij} = -\alpha$ if the corresponding pixels are neighbors, zero otherwise.

The system can be in principle solved by standard methods, e.g. Gaussian elimination, LU-decomposition etc.

It is however obviously too slow ☹

Discrete domain of definition

The matrix A is **sparse** → iterative methods.

For instance the **Jacobi** method: the matrix is decomposed $A = D + M$ into the diagonal part M and the rest:

$$Ay = x \;\Leftrightarrow\; (D+M)y = x \;\Leftrightarrow\; Dy = x - My \;\Leftrightarrow\; y = D^{-1}(x - My)$$

→ the iterative procedure:

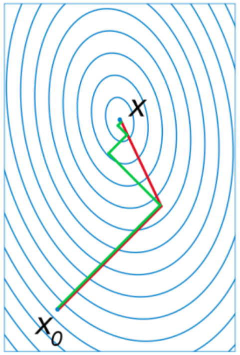
$$y^{(k+1)} = D^{-1}(x - My^{(k)})$$

☺ extremely simple, parallelizable

☹ still too slow, converges at $k \to \infty$ but only if the matrix is strictly diagonal dominant, i.e. $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ (fortunately, just the case here)

Other methods

Conjugate gradients: better convergence is achieved by an appropriate choice of the gradient direction



Successive over-relaxation (special case – Gauss-Seidel method): faster convergence by appropriately chosen gradient step

$$x_k^{(m+1)} = (1-\omega)x_k^{(m)} + \frac{\omega}{a_{kk}}\left(b_k - \sum_{i>k} a_{ki}x_i^{(m)} - \sum_{i<k} a_{ki}x_i^{(m+1)}\right), \quad k = 1, 2, \ldots, n.$$

Multi-grid methods: the domain is coarsened (downscaled), the solution is done very fast → serves as initialization for the more detailed resolution levels (much faster but complicated),

etc.

Continuous domain of definition

$R \subset \mathbb{R}^2$, the mapping $y : \mathbb{R}^2 \to \mathbb{R}$ is a function.

The energy becomes an **energy functional** $E : \mathbb{R}^\infty \to \mathbb{R}$

Example – again the denoising:

$$E(y) = \int_R \Big[\big(y(r) - x(r) \big)^2 + \alpha |\nabla y(r)|^2 \Big] dr \to \min_y$$

data term + model term (smoothness – gradients are penalized)

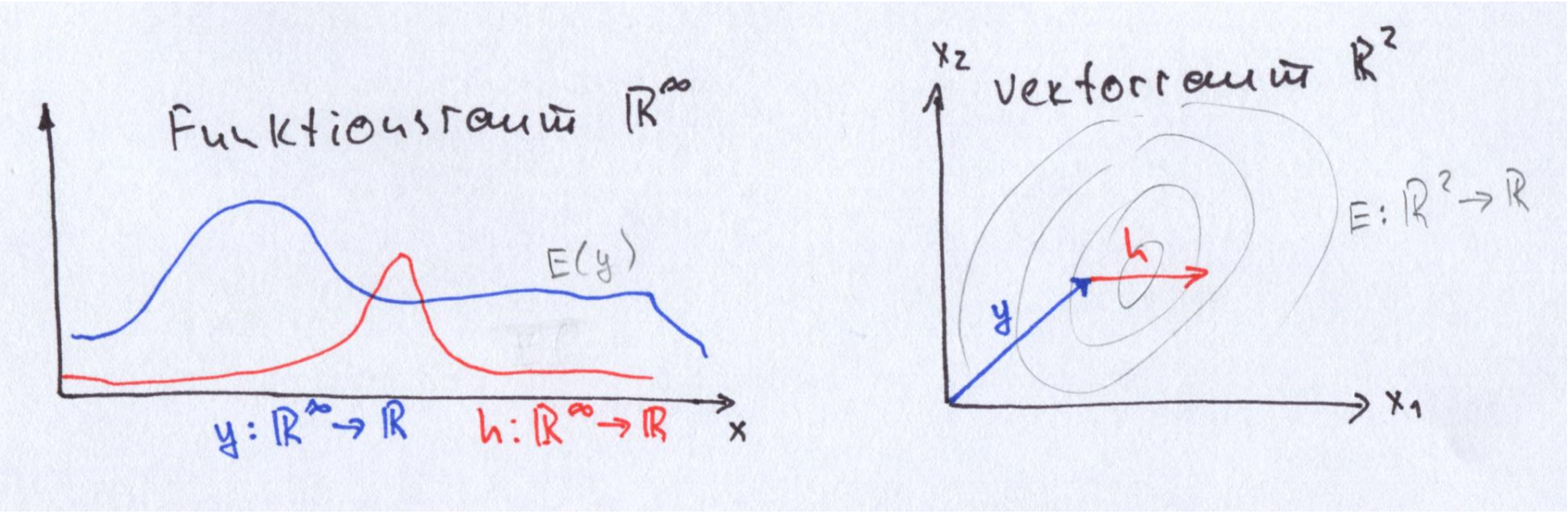
The “problem” – how to derive?

The framework: **Calculus of variations**

Calculus of variations

Gâteaux-derivative:
a generalization of the **directional derivative** on function spaces, "direction" $h : \mathbb{R}^2 \to \mathbb{R}$ is a function too.

$$\frac{\partial E_h(y)}{\partial y} = \lim_{\varepsilon \to 0} \frac{E(y + \varepsilon h) - E(y)}{\varepsilon} = \frac{dE(y + \varepsilon h)}{d\varepsilon}\Big|_{\varepsilon=0}$$



Euler-Lagrange equations: in the optimum all Gâteaux-derivatives (i.e. for all h) are zero.

Gâteaux-derivatives

$$\frac{d}{d\varepsilon}\int_R \left[(y+\varepsilon h-x)^2+\alpha|\nabla(y+\varepsilon h)|^2\right]dr\Big|_{\varepsilon=0} =$$

// koordinatenweise in $\mathbb{R}^2$

$$\frac{d}{d\varepsilon}\int_R \left[(y+\varepsilon h-x)^2+\alpha\Big(\frac{\partial}{\partial r_1}(y+\varepsilon h)\Big)^2+\alpha\Big(\frac{\partial}{\partial r_2}(y+\varepsilon h)\Big)^2\right]dr\Big|_{\varepsilon=0} =$$

$$2\int_R \left[(y+\varepsilon h-x)h+\alpha\Big(\frac{\partial}{\partial r_1}(y+\varepsilon h)\frac{\partial h}{\partial r_1}\Big)+\alpha\Big(\frac{\partial}{\partial r_2}(y+\varepsilon h)\frac{\partial h}{\partial r_2}\Big)\right]dr\Big|_{\varepsilon=0} =$$

$$2\int_R \left[(y-x)h+\alpha\Big(\frac{\partial y}{\partial r_1}\frac{\partial h}{\partial r_1}\Big)+\alpha\Big(\frac{\partial y}{\partial r_2}\frac{\partial h}{\partial r_2}\Big)\right]dr =$$

// partielle Integration

$$2\int_R \left[(y-x)h-\alpha\Big(\frac{\partial^2 y}{\partial r_1^2}h\Big)-\alpha\Big(\frac{\partial^2 y}{\partial r_2^2}h\Big)\right]dr+\ldots(\text{Grenzeffekte}) =$$

$$2\int_R (y-x-\alpha\triangle y)h\ dr+\ldots(\text{Grenzeffekte})=0\quad\forall h$$

$\Downarrow$

$$\Rightarrow\quad y-x-\alpha\triangle y=0\ \ \forall r\in R$$

Comparison with the discrete domain

Euler-Lagrange equations:

$$y - x - \alpha \triangle y = 0 \ \ \forall r \in R$$

Let us discretize it:

$$y_{i,j} - x_{i,j} - \alpha\Big((y_{i-1,j} - 2y_{i,j} + y_{i+1,j}) + (y_{i,j-1} - 2y_{i,j} + y_{i,j+1})\Big) = 0$$

→ the same system of linear equations:

$$y_{i,j}(1+4\alpha) - \alpha y_{i-1,j} - \alpha y_{i+1,j} - \alpha y_{i,j-1} - \alpha y_{i,j+1} = x_{i,j} \quad \forall (i,j).$$

Other discretization schemes, other model-terms

→ other systems

Comparison with diffusion

Gradient descent method to minimize the energy $E(y)$:

$$y^{(t+1)} = y^{(t)} - \frac{\partial E(y)}{\partial y} = y^{(t)} + \alpha \triangle y + (x - y)$$

Compare with the homogenous diffusion:

$$u^{(t+1)} = u^{(t)} + \frac{\partial u}{\partial t} = u^{(t)} + c \triangle u$$

Very similar, up to the term that keeps the solution close to the original image.

Extensions

$$E(y) = \int_R \Big[(y-x)^2 + \alpha \Psi(|\nabla y|^2) \Big] dr \to \min_y$$

with a regularizer Ψ:

$\Psi(s^2) = s^2$ – Tikhonov

$\Psi(s^2) = \sqrt{s^2}$ – Total Variation

$\Psi(s^2) = 1 - \lambda^2 \exp(-\frac{s^2}{\lambda^2})$ – Perona-Malik

$\Psi(s^2) = \begin{cases} 0 & \text{wenn } s^2 = 0 \\ 1 & \text{else} \end{cases}$ – Potts model

Euler-Lagrange equations (non-linear in general):

$$\operatorname{div}\Big(\Psi'(|\nabla y|^2)\nabla y\Big) - \frac{y-x}{\alpha} = 0$$

Summary

Energy Minimization is a sound way to model and solve Computer Vision tasks – they are casted as optimization problems.

(Almost) no hidden assumptions, transparent formulations.

The considered example (denoising) is very simple: **quadratic** penalizer → system of **linear** equations → approaches are very similar to each other and the solution is simple as well.

In general, the problem is “easy” if the subject is **convex**.

Today: continuous energy minimization
Next time: discrete energy minimization (both range and domain)