Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Seminar Notes

Optimisation for Visual Computing Summer Semester 2009

Dr. Michael Breuß,
Dr. Bernhard Burgeth

2009-7-31

Contents

0.1 Euler-Lagrange Formalism

The Swiss mathematician Leonhard Euler and the Italo-French mathematician Joseph-Louis Lagrange (see also figure 0.1) developed the Euler-Lagrange theory in the 1750s. Actually motivated from mechanics, it nowadays pops up in many different fields, e.g. field theory or functional minimisation. Also, there exist connections to iterative algorithms like Gradient Descent or Newton's Method.

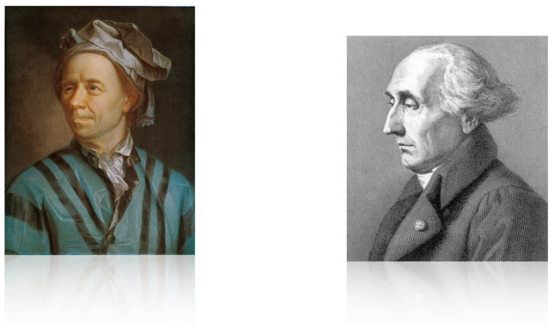

Figure 0.1: **Left:** Leonhard Euler (1707-1783). **Right:** Joseph-Louis Lagrange (1736-1813). **Source:** Wikipedia.

0.1.1 Basic Observations

Besides the standard calculus, which we know from school, there also exists the calculus of variations. For both concepts, let's briefly sketch what they have in common and what is different for them.

Standard Calculus
In standard calculus, one considers real-valued functions $f : \mathbb{R}^N \to \mathbb{R}; (\xi_1, \dots, \xi_N) \in \mathbb{R}^N$. We know from school that f has a minimum in a point ξ if its first derivative $f'(\xi) = 0$ which is the necessary condition for a minimiser (1-D case). Furthermore, if f is strictly convex and $f'(\xi) = 0$, then ξ is a unique minimiser of f.

Calculus of Variations
The calculus of variations considers real-valued "functions" that do not take one or several variables as arguments, but that take a function as argument. In this case, one talks about *functionals*. Such a functional $\mathcal{E} : \mathcal{F} \to \mathbb{R}$ is minimised by a function u if u satisfies the so-called *Euler-Lagrange equations*. This is – like the first derivative in standard calculus – a necessary condition for a minimiser. Similarly, if $\mathcal{E}$ is strictly convex and satisfies the Euler-Lagrange equations, then u is a unique minimiser of $\mathcal{E}$.

Functionals
For the rest of this paragraph, we want to consider functionals that have the following structure

$$\mathcal{E}(u) = \int_a^b \underbrace{(u - f)^2}_{\text{data term}} + \alpha \underbrace{\Psi_S(|\nabla u|^2)}_{\text{smoothness term}} \, d\mathbf{x} \tag{0.1}$$

where u is the (unknown) function for which we minimise, α is a smoothness weight which controls the influence of the smoothness term and Ψ_S is a regulariser that penalises deviations from smoothness. Please note that one can also use a penaliser function Ψ_D to robustify the data term against outliers. The concept of penalising deviations from (measured) data was also part of "Approximation and Fitting I" that was presented by Laurent Hoeltgen.

0.1.2 Euler-Lagrange equations

Let's consider the 1-D and the 2-D case of the Euler-Lagrange theorem.

Theorem (1-D case)

A *smooth function* $u(x)$ with $x \in [a,b]$ that minimises the 1-D energy functional

$$\mathcal{E}(u) = \int_a^b \mathcal{L}(x, u, u') \, \mathrm{d}x \tag{0.2}$$

satisfies necessarily the Euler-Lagrange equation

$$\mathcal{L}_u - \frac{\mathrm{d}}{\mathrm{d}x} \mathcal{L}_{u'} = 0 \tag{0.3}$$

and the so-called natural boundary conditions

$$\mathcal{L}_{u'} = 0 \tag{0.4}$$

for the boundary ($x = a$ and $x = b$) where $\mathcal{L}$ is sometimes called the Lagrangian. Informally speaking, *smooth function* means that one "doesn't get into trouble" when making computations, or more formally, the function u should be as often differentiable as needed.

Theorem (2-D case)

The 2-D case is a simple extension of the 1-D scenario. In principle, u takes two variables as arguments, x_1 and x_2. This requires to use partial derivatives instead of "normal" ones. The theorem reads as follows:
A minimiser of the 2-D energy functional

$$\mathcal{E}(u) = \int_\Omega \mathcal{L}(x_1, x_2, u, u_{x_1}, u_{x_2}) \, \mathrm{d}\mathbf{x} \tag{0.5}$$

satisfies necessarily the Euler-Lagrange equation

$$\mathcal{L}_u - \frac{\partial}{\partial x_1} \mathcal{L}_{u_{x_1}} - \frac{\partial}{\partial x_2} \mathcal{L}_{u_{x_2}} = 0 \tag{0.6}$$

with the natural boundary conditions

$$\mathbf{n}^\top \begin{pmatrix} \mathcal{L}_{u_{x_1}} \\ \mathcal{L}_{u_{x_2}} \end{pmatrix} = 0 \tag{0.7}$$

at the image boundaries $\partial\Omega$ with the normal vector $\mathbf{n}$.

Example for a Variational Problem

Let's give a short example that illustrates how the Euler-Lagrange equations can be used. The task is to find the shortest plane curve joining two points A and B.

One can reformulate this as a variational problem as follows:
Find the curve $u(x)$ for which the functional

$$\mathcal{E}(u(x)) = \int_a^b \sqrt{1 + \left(\frac{\mathrm{d}}{\mathrm{d}x} u(x)\right)^2} \, \mathrm{d}x \tag{0.8}$$

achieves its minimum where $a \leq x \leq b$.
We know that the solution to this problem is a straight line segment joining A and B. The explanation can be given using the Euler-Lagrange formalism. We have the Lagrangian

$$\mathcal{L} = \sqrt{1 + \left(\frac{\mathrm{d}}{\mathrm{d}x} u(x)\right)^2}, \tag{0.9}$$

and we need the following ingredients where the more convenient notation u' is used to indicate the derivative of u with respect to x:

$$\begin{aligned} \mathcal{L}_u &= 0 && (0.10) \\ \mathcal{L}_{u'} &= \frac{1}{2 \cdot \sqrt{1 + (u')^2}} \cdot 2 \cdot u' \\ &= \frac{u'}{\sqrt{1 + (u')^2}} && (0.11) \end{aligned}$$

We can now easily see that u' has to be a constant to satisfy the Euler-Lagrange equation

$$\frac{\mathrm{d}}{\mathrm{d}x} \frac{u'}{\sqrt{1 + (u')^2}} = 0. \tag{0.12}$$

u' being a constant means in other words that $u(x)$ has to be a straight line.

0.1.3 First and Second Variation of an Energy Functional

So far, we assumed that we are able to compute the derivatives $\mathcal{L}_u$, $\mathcal{L}_{u_{x_1}}$ and $\mathcal{L}_{u_{x_2}}$. The problem in this case is that u and its derivatives are functions, not values. But how do we compute a derivative with respect to a function? To answer this question, let's first introduce some notation where we distinguish the continuous and the discrete setting. In the continuous setting, we refer to a functional by using $\mathcal{E}$. The functional $\mathcal{E}$ takes functions u as arguments. In the discrete setting, instead, we have a function E that takes several variables – represented by a vector $\mathbf{u}$ – as arguments.
Now we have the basic ingredients to answer the question how to compute a derivative with respect to a function. The goal is to construct "something" of which we can compute the derivative. This is achieved by actually two steps: a function that brings us to the function space $\mathcal{F}$ and a function that brings us back to $\mathbb{R}$:

- $\mathcal{S} : \mathbb{R} \to \mathcal{F};\ \varepsilon \mapsto u + \varepsilon v$ brings us to the function space $\mathcal{F}$.
- $\mathcal{E} : \mathcal{F} \to \mathbb{R}$ brings us back to $\mathbb{R}$.

$\Rightarrow\ \mathcal{P} : \mathbb{R} \to \mathbb{R};\ \varepsilon \mapsto \mathcal{E}(u + \varepsilon v)$.

Let's think of u and v being vectors $\mathbf{u}$ and $\mathbf{v}$, as we do for the discrete setting. In this case we can consider the set of points

$$\{\mathbf{u} + \varepsilon \mathbf{v} | \varepsilon \in \mathbb{R}\} \tag{0.13}$$

which describes a straight line along the vector $\mathbf{v}$ through $\mathbf{u}$. The derivative of

$$\varepsilon \mapsto E(\mathbf{u} + \varepsilon \mathbf{v}) \tag{0.14}$$

at $\varepsilon = 0$ is described by

$$\frac{\partial}{\partial \varepsilon} E(\mathbf{u} + \varepsilon \mathbf{v})\Big|_{\varepsilon=0}. \tag{0.15}$$

The latter is called directional derivative. Equation (0.15) and

$$\frac{\partial^2}{\partial \varepsilon^2} E(\mathbf{u} + \varepsilon \mathbf{v})\Big|_{\varepsilon=0} \tag{0.16}$$

can also be formulated in the continuous setting, resulting in the first and second variation, respectively:

$$\frac{\partial}{\partial \varepsilon} \mathcal{E}(u + \varepsilon v)\Big|_{\varepsilon=0}, \tag{0.17}$$

$$\frac{\partial^2}{\partial \varepsilon^2} \mathcal{E}(u + \varepsilon v)\Big|_{\varepsilon=0}. \tag{0.18}$$

The intuition should tell us that the first variation is something like a first derivative and the second variation something like a second derivative. Since we are heading for iterative algorithms, we consider from now on only the discrete case. This discrete version of the first variation is given by

$$\begin{aligned} \mathbf{D}E(\mathbf{u})\mathbf{v} &= \frac{\partial}{\partial \varepsilon} E(\mathbf{u} + \varepsilon \mathbf{v})\Big|_{\varepsilon=0} \\ &= \nabla E(\mathbf{u}) \cdot \mathbf{v}, \end{aligned} \tag{0.19}$$

and of the second variation by

$$\begin{aligned}
\mathbf{D}^2 E(\mathbf{u})\mathbf{v} &= \frac{\partial^2}{\partial\varepsilon^2} E(\mathbf{u}+\varepsilon\mathbf{v})\Big|_{\varepsilon=0} \\
&= \frac{\partial}{\partial\varepsilon}(\nabla E(\mathbf{u}+\varepsilon\mathbf{v})\cdot\mathbf{v})\Big|_{\varepsilon=0} \\
&= \frac{\partial}{\partial\varepsilon}\left(\left(\frac{\partial}{\partial u_1}E(\mathbf{u}+\varepsilon\mathbf{v}),...,\frac{\partial}{\partial u_N}E(\mathbf{u}+\varepsilon\mathbf{v})\right)\cdot\mathbf{v}\right)\Big|_{\varepsilon=0} \\
&= \left(\left(\left(\frac{\partial^2}{\partial u_1\partial u_i}E(\mathbf{u})\right),...,\left(\frac{\partial^2}{\partial u_N\partial u_i}E(\mathbf{u})\right)\right)_{i=1,...,N}\cdot\mathbf{v}\right)\cdot\mathbf{v} \\
&= \left(\left(\underbrace{\left(\frac{\partial^2}{\partial u_j\partial u_i}E(\mathbf{u})\right)}_{\mathcal{H}E(\mathbf{u})}\right)_{i,j=1,..,N}\cdot\mathbf{v}\right)\cdot\mathbf{v} \\
&= \mathbf{v}^\top\mathcal{H}E(\mathbf{u})\mathbf{v}. \qquad (0.20)
\end{aligned}$$

This can now be used in iterative algorithms.

0.1.4 Appearance of the Euler-Lagrange Formalism

In this part we are interested in iterative algorithms where the Euler-Lagrange formalism comes into play. For this purpose, we recall

$$\nabla E(\mathbf{u}) = \left(\frac{\partial}{\partial u_j}(E(\mathbf{u}))\right)_{j=1,...,N}, \qquad (0.21)$$

$$\mathcal{H}E(\mathbf{u}) = \left(\frac{\partial^2}{\partial u_j\partial u_i}(E(\mathbf{u}))\right)_{i,j=1,...,N}. \qquad (0.22)$$

Gradient Descent Algorithm

The intuition for the Gradient Descent Algorithm is simple: $\nabla E(\mathbf{u})$ itself points always in the direction of the steepest ascent where $-\nabla E(\mathbf{u})$ obviously points in the opposite direction, i.e. the steepest descent. Since we want to do a minimisation, $-\nabla E(\mathbf{u})$ gives us the search direction for each iteration step of the algorithm:

```
give a starting point $x$
repeat
  $\mathbf{d} = -\nabla E(\mathbf{u})$
  Choose step size $\gamma$ (e.g. by backtracking line search)
  $\mathbf{x} := \mathbf{x} + \gamma \cdot \mathbf{d}$
until stopping criterion is satisfied
```

The algorithm can also be illustrated as shown in figure 0.2.

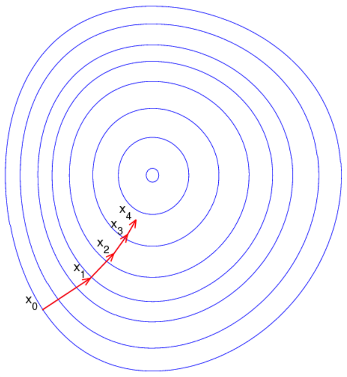

Figure 0.2: Illustration of gradient descent. **Source:** Wikipedia.

Newton's Method

We also want to briefly have a look at Newton's Method:

```
give a starting point $x$
repeat
    $\mathbf{d} = -[\mathcal{H}E(\mathbf{u})]^{-1}\nabla E(\mathbf{u})$
    $\mathbf{x} := \mathbf{x} + \mathbf{d}$
until stopping criterion is satisfied
```

It is important to mention that in a optimisation problem a strictly convex function is desirable. This notion is equivalent to a positive definite Hessian matrix. But what can we do if the Hessian is not positive definite? A solution to this problem could be to perturb the Hessian until it is positive definite which is done in the Trust-Region Method.

0.1.5 Connection to the Euler-Lagrange Formalism

We will now have a look at the connection between the iterative algorithms briefly mentioned above and the Euler-Lagrange formalism. For this purpose we assume the 1-D version of (0.1), i.e.

$$\mathcal{E}(u) = \int_a^b (u-f)^2 + \alpha\Psi_S((u')^2)\,\mathrm{d}x \tag{0.23}$$

This functional is obviously continuous. The question is now how to get a discrete version for the usage in iterative algorithms out of it. Two approaches are possible:

- First employ the necessary optimality condition, then discretise. (OD)
- First discretise, then employ the necessary optimality condition. (DO)

Necessary Optimality Condition → Discretise: OD

First employing the necessary optimality conditions, i.e. employing the Euler-Lagrange equation, gives

$$\begin{aligned} 0 &= 2(u-f) - \frac{\mathrm{d}}{\mathrm{d}x}\left[2\alpha\Psi_S'\big((u')^2\big)\cdot u'\right] \\ &= u - f - \alpha\frac{\mathrm{d}}{\mathrm{d}x}\left[\Psi_S'\big((u')^2\big)\cdot u'\right] \end{aligned} \tag{0.24}$$

After discretisation, we end up in

$$\begin{aligned} 0 &= u_j - f_j - \alpha\frac{\mathrm{d}}{\mathrm{d}x}\left[\Psi_S'\big((u')^2\big)\cdot u'\right]\Big|_{x=x_j} \\ &= u_j - f_j - \alpha\Psi_S'\left(\frac{(u_{j+1}-u_j)^2}{\Delta x^2}\right)\cdot\frac{u_{j+1}-u_j}{\Delta x^2} \\ &\quad +\alpha\Psi_S'\left(\frac{(u_j-u_{j-1})^2}{\Delta x^2}\right)\cdot\frac{u_j-u_{j-1}}{\Delta x^2} \qquad ,j=1,...,N. \end{aligned} \tag{0.25}$$

Discretise → Necessary Optimality Condition: DO

First discretising the energy functional gives

$$E(\mathbf{u}) = \sum_{j=1}^{N}(u_j-f_j)^2 + \alpha\Psi_S\Big((\mathbf{u}')^2\Big)\Big|_{x=x_j} \tag{0.26}$$

After that, wen can employ the necessary conditions for a minimiser, i.e. all partial derivatives must be 0:

$$\begin{aligned} 0 &= \frac{\partial}{\partial u_j}\left(E(\mathbf{u})\right) \\ &= u_j - f_j - \alpha\Psi_S'\left(\frac{(u_{j+1}-u_j)^2}{\Delta x^2}\right)\cdot\frac{u_{j+1}-u_j}{\Delta x^2} \\ &\quad +\alpha\Psi_S'\left(\frac{(u_j-u_{j-1})^2}{\Delta x^2}\right)\cdot\frac{u_j-u_{j-1}}{\Delta x^2} \qquad ,j=1,...,N. \end{aligned} \tag{0.27}$$

As one can easily verify, (0.25) and (0.27) are exactly the same.

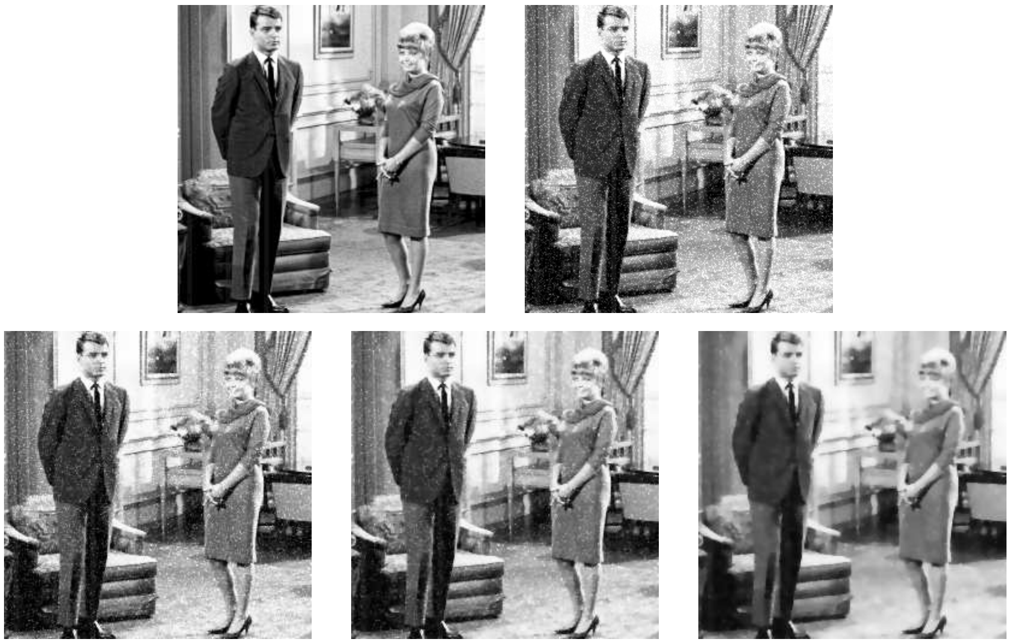

Figure 0.3: **Top Left:** Original image. **Top Right:** Noisy version with Gaussian noise ($\sigma = 40$). **Bottom row:** Denoised version with $\lambda = 1, 2, 5$. The higher the contrast parameter λ, the more noise is removed, but the image is also more blurred.

0.1.6 Examples in Image Processing

In the following, we want to have a look at two different examples from image processing: removing Gaussian noise and removing salt-and-pepper noise.

Removing Gaussian Noise

For this example, let's consider again the energy functional

$$\mathcal{E}(u) = \int_a^b (u - f)^2 + \alpha \Psi_S(|\nabla u|^2)\, \mathrm{d}\mathbf{x} \tag{0.28}$$

with the Charbonnier regulariser

$$\Psi_S(|\nabla u|^2) = 2(\lambda^2 \sqrt{1 + |\nabla u|^2/\lambda^2} - \lambda^2) \tag{0.29}$$

For the further computation, OD was applied to discretise (0.28) for the usage in an iterative algorithm with explicit time stepping where further details are omitted. Let's compare the results after 20 iterations which are given in figure 0.3.

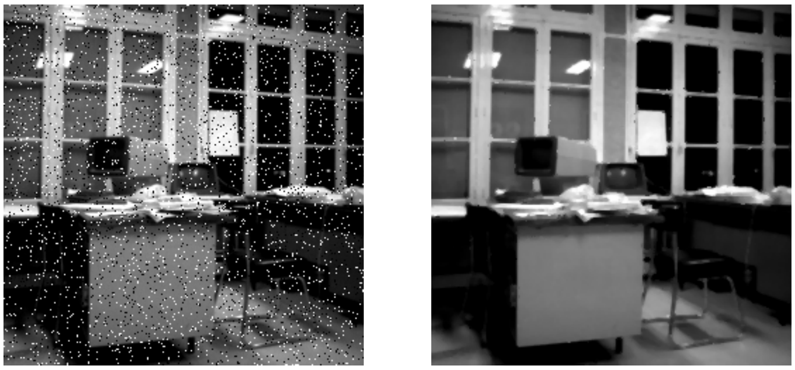

Figure 0.4: **Left:** Noisy version with salt-and-pepper noise. **Right:** Denoised version using the regularised data term $\Psi_D(s^2) = 2(\sqrt{s^2+\varepsilon^2} - \varepsilon)$ and a Charbonnier regulariser as smoothness term $\Psi_S(s^2) = 2(\lambda^2\sqrt{1+s^2/\lambda^2} - \lambda^2)$. **Author:** J. Weickert.

Removing Salt-and-Pepper Noise

A second example is the removal of salt-and-pepper noise given in figure 0.4. This example is taken from the DIC lecture.

0.1.7 Summary

We have seen that the Euler-Lagrange equations are a necessary condition for a minimiser of an energy functional. They are used in several approaches like iterative algorithms. There we have seen the Gradient Descent Algorithm and Newton's Method which are suitable to find a local extremum. For those algorithms, the Euler-Lagrange equations provide the search direction **d** (cf. figure 0.5). Furthermore, we have seen that it doesn't matter whether we employ first

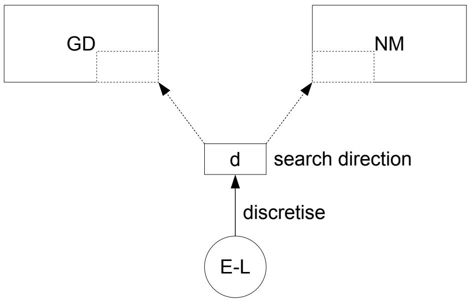

Figure 0.5: The Euler-Lagrange equations provide the search direction for either the Gradient Descent Algorithm or Newton's Method.

the necessary optimality condition – i.e. the Euler-Lagrange equations – and then discretise or vice versa. Examples showed us that energy functional minimisation is a possibility to perform image denoising.

0.1.8 References

- Pablo Pedregal. Introduction to Optimization. Springer New York. 2004.
- I.M. Gelfand, S.V. Fomin. Calculus of Variations. Dover New York. 2000.
- Euler-Lagrange equations (Wikipedia).
  `http://en.wikipedia.org/wiki/Euler-Lagrange_equation`
- Lecture "Differential Equations in Image Processing and Computer Vision". SS 2008.
- Lecture "Numerical Algorithms for Visual Computing III". SS 2009.
- L. Bar, G. Sapiro. Generalized Newton-Type Methods for Energy Formulations in Image Processing. SIAM Journal on Imaging Sciences, 2009.
  `http://www.ece.umn.edu/users/barxx002/papers/Siam09.pdf`