R. Herzog, G. Müller
Universität Heidelberg

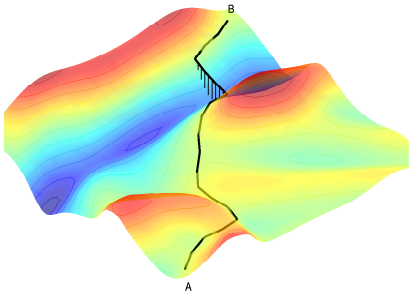
Exercise 1 - Solution

Problem 1.1. (Railway planning problem)

Given two points A and B in $\mathbb{R}^3$ (that are reasonably close), we are looking to construct a railway connecting A and B at the lowest cost possible. The topography of the region around A and B is described by a function $z\colon \mathbb{R}^2 \to \mathbb{R}$ mapping a point $(x, y) \in \mathbb{R}^2$ to the vertical height above sea level $z(x, y)$ at (x, y), i. e., the points on the surface of the earth are described by $(x, y, z(x, y))$. We need to avoid steep inclines along the new railway, a requirement that we simplify by assuming that A and B are at sea level ($A_z = B_z = 0$) and requiring that the entire railway remains at sea level as well. To achieve this, we may build tunnels and bridges through/over topographic features, both of which cost money depending on their depth/height. The total cost for a piece of railway of infinitesimal length ds at the position $(x, y, z(x, y))$ is given by

$$(C_1 + C_2\, z(x, y)^2)\ \mathrm{d}s$$

for constants $C_1, C_2 > 0$.



Topography around A and B with tunnels and bridges.

Model the problem as an optimization problem of the form (1.1) in the lecture notes that can be solved numerically and approximately solve the problem for the parameters

- $A = (-1, -1, 0)^\mathsf{T}$, $B = (1, 1, 0)^\mathsf{T}$
- $z(x, y) = \frac{1}{8} \sum_{i=1}^{6} a_i(x, y)$ with

$$\begin{aligned}
a_1(x, y) &= -f(3 - 0.7x^2 - 5y) \\
a_2(x, y) &= f(5(x - 0.7)^2 + 20(y - 0.1)^2) \\
a_3(x, y) &= f(4 + 2x^2 - 4y) \\
a_4(x, y) &= 0.5 \sin(4(x - y^2)) \sin(3(y + x^2)) \\
a_5(x, y) &= 0.6 f(50((x + 1)^2 + (y + 0.8)^2 - 0.25)^2) \\
a_6(x, y) &= -1/6
\end{aligned}$$

  where $f(x) = 2/(1 + x^2)$.

using a suitable numerical solver of your choice. Investigate the effects of the parameters C and the initial guess that you supply to the solver.

Solution.

The optimization variable that we can control in this problem is the path of the railway. It can be described by a path $\gamma\colon [0, 1] \to \mathbb{R}^3$ from A to B that is supposed to satisfy the additional constraint $\gamma_3 \equiv 0$.

The cost for any given path like that can be derived from the infitisimal cost per length from the task description by integrating along the path. Note that the constraint $\gamma_3 \equiv 0$ is already explicitly incorporated in the cost functional (where the cost would otherwise include the difference of z and γ_3).

For now, we therefore obtain the optimization problem

$$\begin{aligned}
\text{minimize} \quad & \int_\gamma C_1 + C_2 \| z(x, y) - \underbrace{\gamma_3}_{=0} \|^2 \, \mathrm{d}s \text{ over } \gamma \in C([0, 1])^3 \\
\text{such that} \quad & \gamma_3 \equiv 0 \\
& \gamma(0) = A = (A_1, A_2, 0) \\
& \gamma(1) = B = (B_1, B_2, 0)
\end{aligned}$$

which we can modify to only consider twodimensional curves by dropping the altitude constraint of γ_3, since it has no effect, i. e., we obtain

$$\begin{aligned} \text{minimize} \quad & \int_\gamma C_1 + C_2 \|z(x,y)\|^2 \,\mathrm{d}s \text{ over } \gamma \in C([0,1])^2 \\ \text{such that} \quad & \gamma(0) = (A_1, A_2) \\ & \gamma(1) = (B_1, B_2). \end{aligned}$$

Since the space $C([0,1])^2$ is infinite dimensional, we need to apply some sort of discretization to obtain an optimization problem that we can deal with numerically. While there are more elegant ways of discretizing curves (see, e. g., Bezier curves), we go the simples route possible by choosing a fixed number N of control points $(x^{(i)}, y^{(i)})$, $i = 1, \dots, N$ on γ and assuming γ to consist of $N-1$ linear segments γ_i, $i = 1, \dots, N-1$ between these control points. We assume that $\gamma^{(i)} := \gamma(t^{(i)}) = (x^{(i)}, y^{(i)})$ for $0 = t^{(1)} \le \cdots \le t^{(n)} = 1$ (meaning that both A and B are also control points for now).

Accordingly, the cost functional of the discretized optimization problem becomes

$$\begin{aligned} \int_\gamma C_1 + C_2 z(x,y)^2 \,\mathrm{d}s &\approx \sum_{k=1}^{N-1} \int_{\gamma_i} C_1 + C_2 z(x,y)^2 \,\mathrm{d}s \\ &= \sum_{k=1}^{N-1} \int_{t^{(k)}}^{t^{(k+1)}} \left[C_1 + C_2 z\left(\gamma^{(k)} + \frac{t - t^{(k)}}{t^{(k+1)} - t^{(k)}} (\gamma^{(k+1)} - \gamma^{(k)}) \right)^2 \right] \|\dot\gamma(t)\| \,\mathrm{d}t \\ &= C_1 \sum_{k=1}^{N-1} \left\| \gamma^{(k+1)} - \gamma^{(k)} \right\| \\ &\quad + C_2 \int_{t^{(k)}}^{t^{(k+1)}} z\left(\gamma^{(k)} + \frac{t - t^{(k)}}{t^{(k+1)} - t^{(k)}} (\gamma^{(k+1)} - \gamma^{(k)}) \right)^2 \left\| \gamma^{(k+1)} - \gamma^{(k)} \right\| \mathrm{d}t. \end{aligned}$$

The optimization variable is now no longer the full path γ but rather the defining "corners" $\gamma^{(k)}$, $k = 1, \dots, N$ of the discretized path (including the start an end point). Accordingly, the reduced discretized optimization problem reads as

$$\begin{aligned} \text{minimize} \quad & C_1 \sum_{k=1}^{N-1} \left\| \gamma^{(k+1)} - \gamma^{(k)} \right\| + C_2 \int_{t^{(k)}}^{t^{(k+1)}} z\left(\gamma^{(k)} + \frac{t - t^{(k)}}{t^{(k+1)} - t^{(k)}} (\gamma^{(k+1)} - \gamma^{(k)}) \right)^2 \left\| \gamma^{(k+1)} - \gamma^{(k)} \right\| \mathrm{d}t \\ \text{over} \quad & \gamma^{(k)},\ k = 2, \dots, N-1 \\ \text{where} \quad & \gamma^{(1)} = A \\ & \gamma^{(N)} = B. \end{aligned}$$

Note that the initial and terminal condition here are not constraints. Rather they are explicit incorporations of the constraints into the problem formulation ($\gamma^{(1)}$ and $\gamma^{(N)}$ are not optimization variables!).

The only remaining issue is the integral, which can typically not be evaluated exactly. We can either apply any reasonable quadrature rule ahead of time in the modeling process or just let an optimizer deal with it using a numerical quadrature routine (if we are not required to supply the derivative).

In case of the trapezoidal rule, the cost functional reads as

$$\begin{aligned} \text{minimize} \quad & C_1 \sum_{k=1}^{N-1} \left\| \gamma^{(k+1)} - \gamma^{(k)} \right\| + C_2 \sum_{k=1}^{N-1} 0.5(z(\gamma^{(k)})^2 + z(\gamma^{(k+1)})^2) \left\| \gamma^{(k+1)} - \gamma^{(k)} \right\| \\ \text{over} \quad & \gamma^{(k)},\ k = 2, \dots, N-1 \text{ with } \gamma^{(1)} = A \text{ and } \gamma^{(N)} = B. \end{aligned}$$

As sated, the optimization variable in numerically solving will be the x_i and y_i of $\gamma_i = (x_i, y_i)$ concatenated to yield a $\mathbb{R}^{2N-2}$ vector.

Either way, the problem has to be considered an unconstrained but fully nonlinear problem, as the topography of the landscape is highly irregular and this is not only a sum of squared residuals, hence nonlinear solvers such as (Quasi-)Newton type methods are a reasonable option. Preimplemented version can be found in Matlab's *fminunc* and in Python, one can turn to SciPy's optimization toolbox.

Please, take a look at the supplied solution code and feel free to investigate some of the effects.

Some things to try out:

(i) In the numerical results for the trapezoidal-rule-version of the problem, we observe that the solver will simply return solutions that have all support points sitting on the same levelset of the topography but the connecting straight edges can pass through very rough terrain (due to the fact that the trapezoidal rule ignores the function z in between the support points). This shows that a higher level quadrature rule (such as a Gauß Quadrature) is required for this problem.

(ii) The results are strongly dependent on the parameters C, as their balance influences, how much the solver focuses on either part of the cost functional. The values of z^2 are quite small compared to the length of the paths, so C_2 needs to be chosen much smaller than C_1 to see some focus on the height difference of the path and the topography. Try $C_1 = 1$, $C_2 = 10^2$ or even $C_2 = 1e-5$.

(iii) The problem is strongly dependent on the initial guess for the solvers. There is a parameter $\alpha \in (0, 1)$ in the code, that controls how much the initial guess connecting A and B is moved

away from the diagonal straight line $[A, B]$ ($\alpha = 0.5$ is the diagonal. Play around and check the difference in the solutions.

(*iv*) The problem is highly nonlinear, giving the solvers issues with local minimizers. You may notice some configurations where the solver simply returns an unuseful and unintuitive solution. It probably got stuck in a local minimum and does not have the same notional of global "usefulness" that the user has. Best you can hope for is that the solver finds a stationary point. If it does, then this might be a local minimizer if you're lucky. Rarely, this will happen to be a global minimizer in fully nonlinear problems such as this. Especially if the space of possible contorl points is large. This will get better, when constraints are added (such as maximum distance between control points) – but we are not equipped to deal with those yet.