Motion Planning — Exercise 11

Wolfgang Hönig and Andreas Orthey, TU Berlin SS 2024

1. Paths and trajectories can be classified according to their smoothness ($C^0, C^1, \ldots, C^\infty$).
   (a) What category of smoothness has the constant function $f(x) = x$?
   (b) Assume that a single robot actuator is operated using a C^3, but not C^4, function. Does this mean that the acceleration of this function is differentiable? What about the jerk function?

2. B-Splines are one tool to smooth paths. However, they typically do not take constraints like joint limits into account. Explain three ways of how you could account for joint limits when smoothing with B-splines.

3. Assume that you ran a sampling based path planner, and you are given a list of points representing a path from $(0, 0)$ to $(1, 1)$. You now want to optimize your path represented by points p1 to p9 using a gradient descent method.
   (a) Visualize the shape of the path in 2D using matplotlib:

```
p0: [0.0,0.0]
p1: [0.1,0.8]
p2: [0.2,—0.8]
p3: [0.3,0.2]
p4: [0.4,0.3]
p5: [0.5,—0.9]
p6: [0.6,0.4]
p7: [0.7,—0.4]
p8: [0.8,0.1]
p9: [0.9,—0.6]
p10: [1.0,1.0]
```

   (b) Let the path length cost of a point p_i ($i = 1 \ldots 9$) be $U_{length}[p_i] = \|p_i - p_{i-1}\|_2 + \|p_i - p_{i+1}\|_2$, i.e., the pathlength of the two line segments that include p_i. Compute the gradient of $U_{length}[p_i]$ with respect to p_i, assuming that both p_{i-1} and p_{i+1} are static.
   (c) Implement the functional gradient descent algorithm presented in the lecture, by iteratively updating individual points p_i for $i = 1 \ldots 9$ and plot the result.
   (d) Define the smoothness cost U_{smooth} as the negative dot product of the normalized vectors $\overrightarrow{p_{i-1}p_i}$ and $\overrightarrow{p_ip_{i+1}}$. As before, compute the gradient and add U_{smooth} to your objective. How does adding this term affect the solution and convergence rate of your algorithm?

   Hint: If you can not compute the gradient analytically, you might consider a autodiff-package such as jax, which comes with a lightweight wrapper for numpy.