C\(^1\) interpolation of locally convex data set

We extend the algorithm for interpolating a convex data set with a C\(^1\)-smooth convex surface [Renka, 2004], so that the algorithm can handle a data set that is only locally convex.

Before we discuss the extension, we review the original algorithm. Given a (strictly) convex data set \(\{(p_i,z_i)\}\), there is a (unique) 2D triangulation of the data points \(\{p_i\}\) on the plane such that the piecewise linear surface over vertices connected by the triangulation is a convex polyhedron. This triangulation is also known as the 2D regular triangulation. After constructing the regular triangulation, the algorithm estimates a functional gradient at each vertex, uses these gradients to determine linear functions at the vertices whose maximum forms a piecewise-linear convex envelope, and finally smoothes the piecewise linear function through convolution with a polynomial function that is locally supported. The convolved function is C\(^1\) smooth. For a detailed description, please refer to [Renka, 2004].

Our extension utilizes the locally supported property of the convolution. Our input is a locally regular triangulation that yields a locally convex piecewise linear surface. We thus avoid the first step from the original algorithm of constructing the regular triangulation. We estimate functional gradients at vertices and do the subsequent processing as before. The local convexity of the piecewise linear function and the local convolution together determine the local convexity of the resulting interpolated C\(^1\)-smooth surface.

We summarize the two procedures for creating the smooth interpolated surface and evaluating this surface at a given point in Alg. 1 and Alg. 2. Our algorithm relies on routines from CSRFPACK [Renka, 2004].

\textbf{Remark.} In order for the algorithms to work, the power cell \(C_i \in \mathcal{P}_g\) must contain its corresponding point \(p_i \in T\). The choice of \(g_i\) influences this. Taking \(g_i\) as the centroid of the power cell, as Alg. 1 does, generally works.

\textbf{Algorithm 1 create_interp_surface(\(\))\(\)}
\begin{itemize}
\item \textbf{Input:} A locally regular triangulation \(T\) with data points \(\{p_i,z_i\}\) as vertices. Weight \(w_i := p_i^2 - 2z_i\).
\item \textbf{Output:} A locally regular triangulation \(T_g\) and its dual power diagram \(\mathcal{P}_g\), the minimum Euclidean distance \(\delta\) from any vertex in \(T\) to the boundaries of its corresponding cell in \(\mathcal{P}_g\).
\end{itemize}

1. Compute the dual power diagram of \(T\), and the centroid \(g_i\) of the power cell corresponding to vertex \(p_i\) is the estimated gradient for \(p_i\).
2. Construct triangulation \(T_g\) of \(\{g_i\}\) using the connectivity of \(\{p_i\}\) in \(T\), assign power weights to \(g_i\): \(w_{g_i} := g_i^2 - 2(g_i, p_i) - z_i\) and make \(T_g\) locally regular through edge flipping.
3. Compute the dual power diagram \(\mathcal{P}_g\) of \(T_g\), and the minimum distance \(\delta\).

\textbf{Algorithm 2 eval_interp_surface(\(\))\(\)}
\begin{itemize}
\item \textbf{Input:} \(T, \mathcal{P}_g, \delta, \) a 2D point \(p\) at which to evaluate the function, and a nearby vertex \(p_0 \in T\) for locating the nearest power cell for \(p\).
\item \textbf{Output:} \(f(p)\) and \(\partial f(p)\).
\end{itemize}

1. Starting from the power cell \(C_0 \in \mathcal{P}_g\) corresponding to \(p_0\), locate the power cell \(C_t \in \mathcal{P}_g\) that contains \(p\) and corresponds to \(p_t \in T\).
2. Compute the distance \(\delta_p\) from \(p_0\) to \(C_t\).
3. \textbf{If} \(\delta_p > \delta\), \textbf{return} \(f(p)\) as the linear function \(H_t(p)\) at \(p_t\), i.e., \(f(p) = H_t(p) = (g_t, p - p_t) + z_t\), \(\partial f(p) = \partial H_t(p) = g_t\); else compute \(f(p)\) through convolution, which is numerically a weighted summation of \(H_{ij}(p + q_{ij})\), where \(\|q_{ij}\| \leq \delta\) is the offset, and the power cell \(C_{ij} \in \mathcal{P}_g\) contains \(p + q_{ij}\). The convolution is done by CSRFPACK.

\textbf{References}