A Polynomial Time Algorithm for Multivariate Interpolation in Arbitrary Dimension via the Delaunay Triangulation

Tyler Chang*, Layne Watson, Thomas Lux, Bo Li, Li Xu, Ali Butt, Kirk Cameron, and Yili Hong

Virginia Polytechnic Institute and State University

March, 2018

* corresponding author: thchang@vt.edu

Table of Contents

Introduction
- What is the Delaunay Triangulation?
- How is it useful?

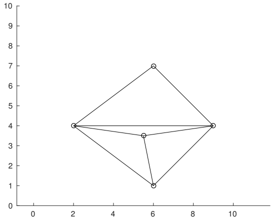
Background
- Formal Definition
- Challenges in High Dimensions
- Previous Works

A New Algorithm
- Addressing challenges
- Necessarry Operations
- The New Algorithm

Results and Conclusion

What is the Delaunay Triangulation

- A *triangulation* of a (finite) set of points P in $\mathbb{R}^d$ is a space filling simplical-mesh, whose elements are disjoint except along shared boundaries, whose vertices are points in P, and whose union is the *convex hull* of P.



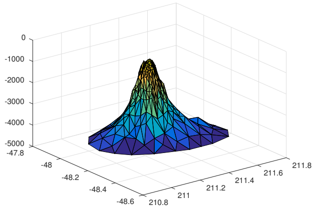
- The *Delaunay triangulation* is a special triangulation, often considered optimal for interpolation purposes.

Uses

- Mesh generation (e.g., for finite element methods)
- Topological data analysis
- Graph theory

Uses

- Mesh generation (e.g., for finite element methods)
- Topological data analysis
- Graph theory
- **Piecewise linear multivariate interpolation**



Interpolation

- Let $T(P)$ be a d-dimensional triangulation of P.
- Let $q \in CH(P)$ be an interpolation point, and let S be a simplex in $T(P)$ with vertices s_1, …, s_{d+1} such that $q \in S$.
- Then there exist unique *convex weights* w_1, …, w_{d+1} such that $q = \sum_{i=1}^{d+1} w_i s_i$.

Define:

$$\hat{f}_T(q) = f(s_1)w_1 + f(s_2)w_2 + \ \dots \ + f(s_{d+1})w_{d+1}.$$

Definition of the Delaunay Triangulation

The Delaunay triangulation is usually defined as the geometric dual of the *Voronoi diagram*. However, the following equivalent definition is preferred:

Definition

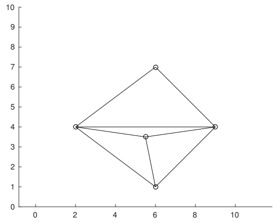
Let $C(v,r)$ denote a sphere of radius r centered at v, and let $B(v,r)$ denote the corresponding open ball:

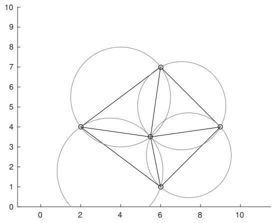
$$C(v,r) := \{x \in \mathbb{R}^d : \|x - v\| = r\}, \quad B(v,r) := \{x \in \mathbb{R}^d : \|x - v\| < r\}.$$

A Delaunay triangulation $DT(P)$ of n points $P \subset \mathbb{R}^d$ is any triangulation of P such that for each d-simplex $S \in DT(P)$, the $(d-1)$-sphere $C(v,r)$ circumscribing S satisfies $B(v,r) \cap P = \emptyset$.

Visual of the Delaunay Triangulation

A triangulation of 5 points in the plane $\mathbb{R}^2$ (left) and the Delaunay triangulation of those same points (right):





Existence and Uniqueness

- Degeneracies:
  - If all n points in P lie in some $(d-1)$-dimensional linear manifold, then no triangulation can *exist*.
  - If $d+2$ or more points lie on the same $(d-1)$-sphere, then dividing these $d+2$ points into 2 or more $(d+1)$-simplices can be done arbitrarilly, and the Delaunay triangulation is not *unique*.
- If neither of the above 2 degeneracies occur, then the points are said to be in *general position*, and the Delaunay triangulation $DT(P)$ exists and is unique.

Computability

- There are many efficient algorithms for computing the two- and three-dimensional Delaunay triangulation (in $\mathbb{R}^2$ and $\mathbb{R}^3$) generally running in $\mathscr{O}(n \log n)$ time.
- However, in $\mathbb{R}^d$ the size of the Delaunay triangulation grows exponentially! In the worst case: $\mathscr{O}(n^{\lceil \frac{d}{2} \rceil})$!
  $\Rightarrow$
  - Requires at least $\mathscr{O}(n^{\lceil \frac{d}{2} \rceil})$ time to compute;
  - Requires at least $\mathscr{O}(n^{\lceil \frac{d}{2} \rceil})$ space to store;

Computability

- There are many efficient algorithms for computing the two- and three-dimensional Delaunay triangulation (in $\mathbb{R}^2$ and $\mathbb{R}^3$) generally running in $\mathcal{O}(n \log n)$ time.
- However, in $\mathbb{R}^d$ the size of the Delaunay triangulation grows exponentially! In the worst case: $\mathcal{O}(n^{\lceil \frac{d}{2} \rceil})$!
  $\Rightarrow$
  - Requires at least $\mathcal{O}(n^{\lceil \frac{d}{2} \rceil})$ time to compute;
  - Requires at least $\mathcal{O}(n^{\lceil \frac{d}{2} \rceil})$ space to store;
- :(

Previous attempts!

Many have tried to crack the curse of dimensionality!

- Bowyer-Watson (Bowyer and Watson 1981)
- Quickhull (Barber, Dobkin, and Huhdanpaa, 1996)
- CGAL — Delaunay graph implementation (Boissonnat, Devillers, and Hornus, 2009)

But ultimately, there's no getting around the exponential nature of the problem...

A potential solution

Proposed solution: Instead of computing the whole triangulation, just compute the part we need!

- Recall the interpolation formula:

$$\hat{f}_T(q) = f(s_1)w_1 + f(s_2)w_2 + \ \ldots \ + f(s_{d+1})w_{d+1}.$$

  Only dependent on a single simplex $S \in DT(P)$ (the simplex containing q)!
- Can we compute S without computing all of $DT(P)$?

Necessarry Operations

There are three necessarry operations:

Necessarry Operations

There are three necessarry operations:

- Grow an initial simplex

Necessarry Operations

There are three necessarry operations:

- Grow an initial simplex
- “Flip” a simplex

Necessarry Operations

There are three necessarry operations:

- Grow an initial simplex
- “Flip” a simplex
- Walk to containing simplex

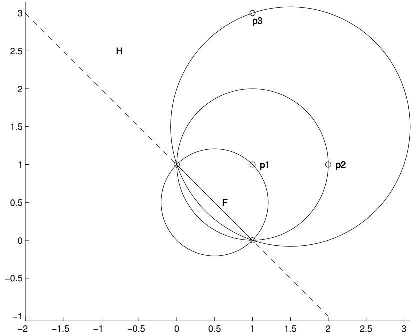
Growing Initial Simplex

Minimize the radius of the initial circumsphere. A greedy algorithm works!

- Pick initial point from P arbitrarilly (closest point to q is a good idea)
- Pick second point from P that is closest to q
- Pick third point to minimize the radius of the smallest circumsphere
- Etc...

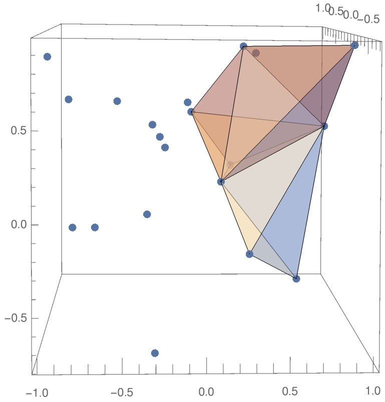
Flipping a Simplex

Drop a point, then pick a new point on the "other side" of the remaining facet to get a new simplex (hopefully "closer" to q)



Simplex Walk

Flip toward the next point using a "visibility walk"



The Algorithm

- Grow an initial simplex
- Flip “toward” q using a visibility walk
- Once we’ve found the simplex containing q, use the interpolation formula on the vertices of the simplex!

Time/Space Complexity

Total runtime:

- $\mathcal{O}(nd^4)$ to compute first simplex
- $\mathcal{O}(nd^3)$ to compute each "flip"
- Total number of flips seems to be $\Theta(d \log d)$:

Table: Average number (with a sample size of 20) of Delaunay simplices computed in a simplex walk for n pseudo-randomly generated points in d dimensions.

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
$d = 2$	3.05	2.90	3.25	3.10
$d = 8$	23.75	24.75	24.30	23.10
$d = 32$	95.25	125.60	131.85	150.10
$d = 64$	171.95	221.85	248.35	280.60

Issues in Stability

- Points could all lie in a $(d-1)$-dimensional linear manifold
  - Nothing to be done: Users should apply dimension reduction techniques
- $d+2$ or more points lie in a $(d-1)$-sphere
  - Delaunay triangulation still exists, but is not unique
  - Still, compute *a* Delaunay triangulation
- A price to be paid! Extra computation time spent handling degeneracies.

Implementation

- A serial numerically stable implementation of the proposed algorithm has been coded in ISO Fortran 2003.
- Tested for correctness against the standard implementation of Quickhull.
- Runtimes gathered for pseudo-randomly generated data on a lab computer:
- Note that Quickhull and other implementations don't scale past moderately sized data sets in more than 5 or 6 dimensions, so there is no comparison.

Results

Table: Average runtime in seconds for interpolating at uniformly distributed interpolation points for n pseudo-randomly generated input points in 5 dimensions.

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
32 interp. pts	0.3 s	2.7 s	9.6 s	35.7 s
1024 interp. pts	2.5 s	11.6 s	28.9 s	79.1 s

Results

Table: Average runtime in seconds for interpolating at clustered interpolation points for n pseudo-randomly generated input points in 5 dimensions.

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
32 interp. pts	0.2 s	2.2 s	8.4 s	33.0 s
1024 interp. pts	0.2 s	2.5 s	9.2 s	35.2 s

Results

Table: Average runtime in seconds for interpolating at a single point for n pseudo-randomly generated input points in d-dimensional space .

	$n = 2K$	$n = 8K$	$n = 16K$	$n = 32K$
$d = 2$	0.1 s	1.7 s	6.8 s	27.0 s
$d = 8$	0.2 s	2.5 s	9.6 s	37.9 s
$d = 32$	1.4 s	9.5 s	29.7 s	101.1 s
$d = 64$	13.2 s	60.1 s	138.6 s	349.1 s

Conclusion and Future Work

By taking advantage of the interpolation problem structure we are able to make this previously exponentially complex problem extremely scalable!

In future work, these same techniques could be applied to other problems that use Delaunay triangulations (including Voronoi diagrams).

QUESTIONS?