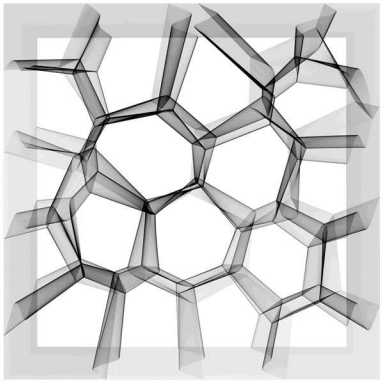
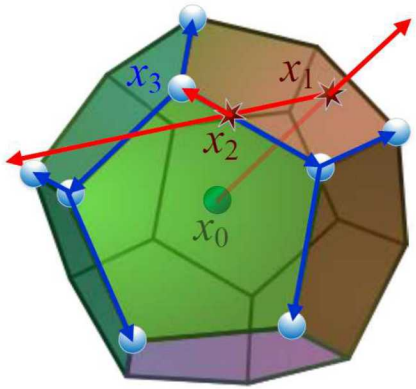
SAND2018-5906C





Solving classification problems using implicit Voronoi cells and local hyperplane sampling

Workshop on Research Challenges and Opportunities at the interface of Machine Learning and Uncertainty Quantification
USC, June 4-6, 2018

Mohamed Ebeida, Laura Swiler, Eric Phipps, Jaideep Ray

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories

Machine learning in Model Calibration

Main use case

Identification of points in input parameter space which are physically feasible and can be used as reliable training data for emulators

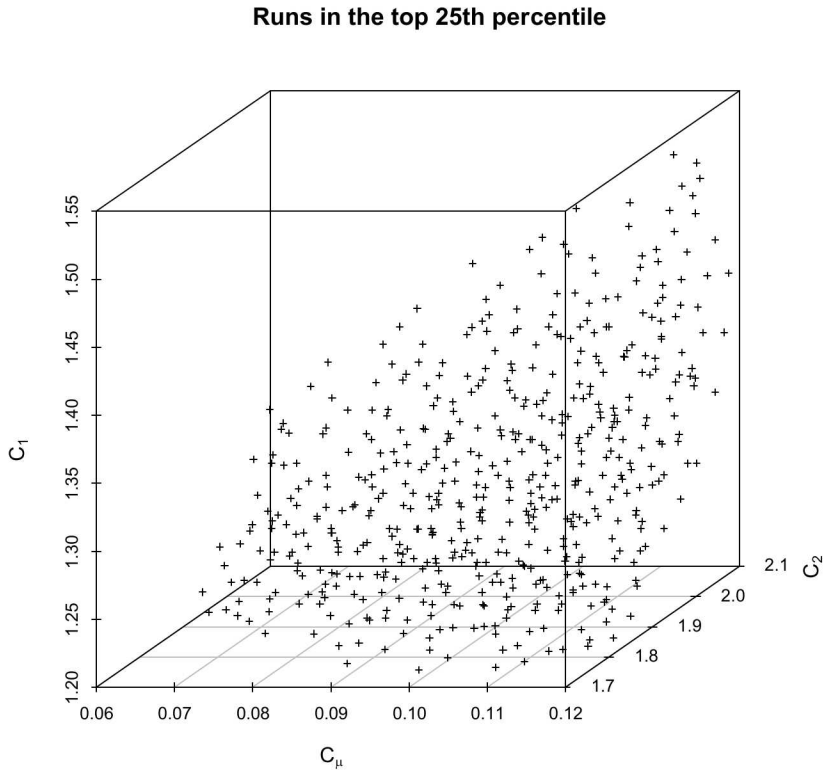
- Emulators are required for Bayesian calibration
- Taking training data over a hypercube defining the parameter bounds is not helpful: many parameter combinations may be infeasible or non-physical
- Goal: if not feasible, do not allow in the prior
  - And don't sample over them in MCMC
- One example where this is needed: turbulence model calibration

The turbulence calibration problem

- Turbulence models for k-ε Reynolds-Averaged Navier-Stokes (RANS) equations contain parameters (C_μ, $C_{\varepsilon,1}$, $C_{\varepsilon,2}$) etc. that need to be calibrated, for the model to be predictive
  - Also have to obey some constraints e.g., $\sigma_\epsilon = \frac{\kappa^2}{\sqrt{C_\mu}(C_{\epsilon,2}-C_{\epsilon,1})}$
  - Also many (C_μ, $C_{\varepsilon,1}$, $C_{\varepsilon,2}$) combinations are non-physical and the code crashes
- Clearly, the parameter space for inference is *not* a hyper-rectangle
  - One can sample the parameter space, crash one's code repeatedly to find valid and invalid points
  - This occurs when we are trying to make emulators to be used in the inverse problem
    - Turbulent flow forward problems are computationally expensive & emulators are a must

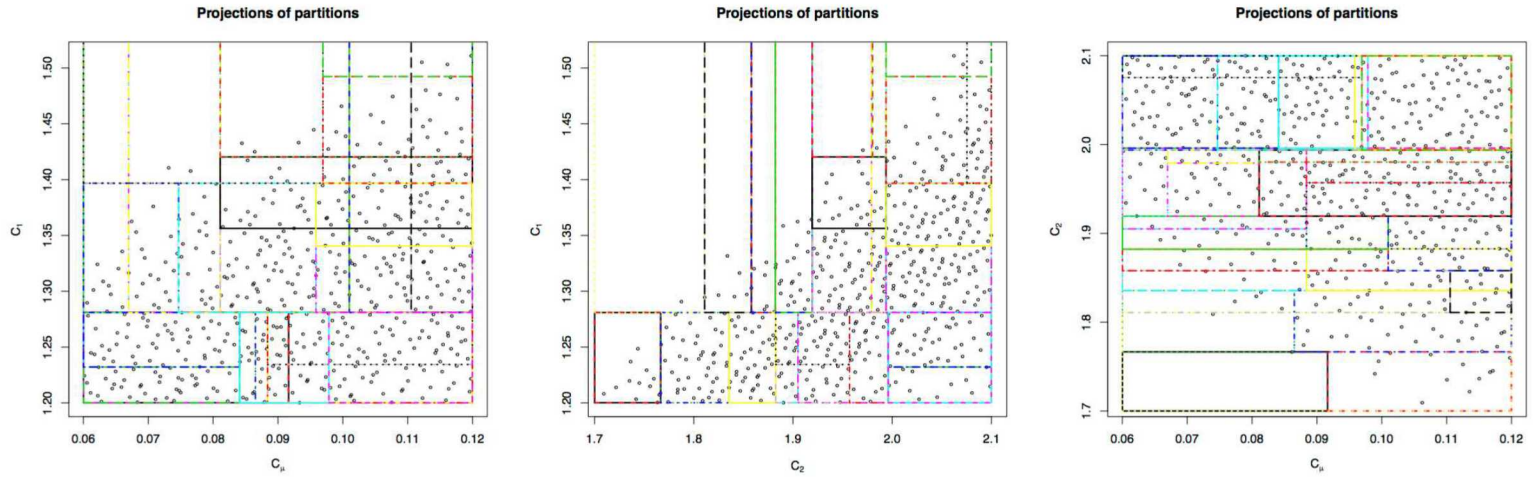
Making an informative prior

- If we are to impose constraints and stay out of non-physical parameter space, we'll need an informative prior
  - So what does the "good" part of the parameter space $\mathcal{R}$ look like?
  - Complex. Can't be analytically defined
- To stay within $\mathcal{R}$ while sampling over (C_μ, $C_{\varepsilon,1}$, $C_{\varepsilon,2}$, κ), we need a binary function $\mathcal{F}(C_\mu, C_{\varepsilon,1}, C_{\varepsilon,2}, \kappa)$
  - In other words, a classifier

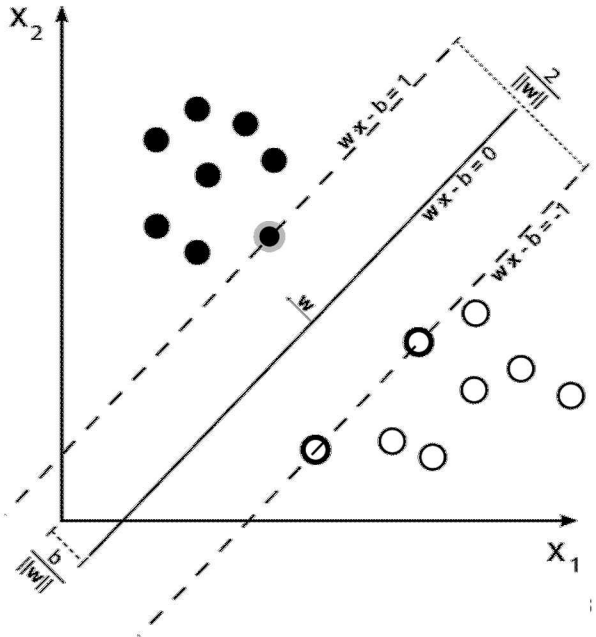


Sandia National Laboratories

How is this done today?

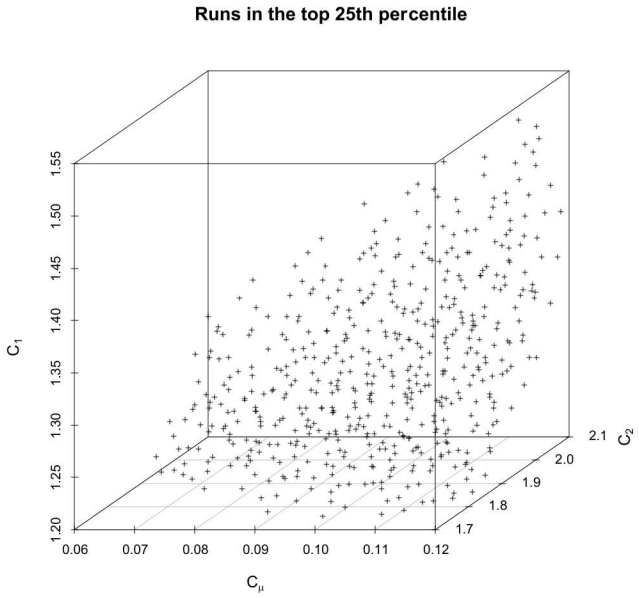


- In Guillas et al, 2014 where $\mathcal{R}$ was relatively simple, they used a rather complex MVN distribution that had $\mathcal{R}$ as support
- In Ray et al, 2015, classification & regression trees were used
- In Ray et al, 2016, they used a SVM



What are the short-comings?

- The shortcoming has the same cause - defining a complex R with few points
  - $\mathcal{R}$ is defined by running an expensive simulator repeatedly, and one can run that only so many times
- Classification trees recursively decompose a rectangular domain into disjoint boxes
  - They resolve a non-rectangular $\mathcal{R}$ approximately by staircasing. Hard to do with few points



- Complex MVN can become intractable unless $\mathcal{R}$ is rather simple
- SVMs are great, but they need a densely populated support vector to resolve complex $\mathcal{R}$
- So why not try a different approach – k-nearest neighbors?

K-Nearest Neighbors

- In this work, we use K-nearest neighbors instead of SVM
- KNN takes a vector of features and predicts to which class it belongs, based on a set of N training data points.
- Each data point is described with a *feature vector* and a *class category.*
- The features may be categorical or numeric values. We assume the classes are discrete.
- The class membership of the new point is predicted by the majority vote of the neighbors.
  - Example: if k=5 and a new point has 3 nearest neighbors in class A and 2 nearest neighbors in class B, the point would be classified as belonging to class A.
- Distance metrics are important: how do we determine “nearest”?

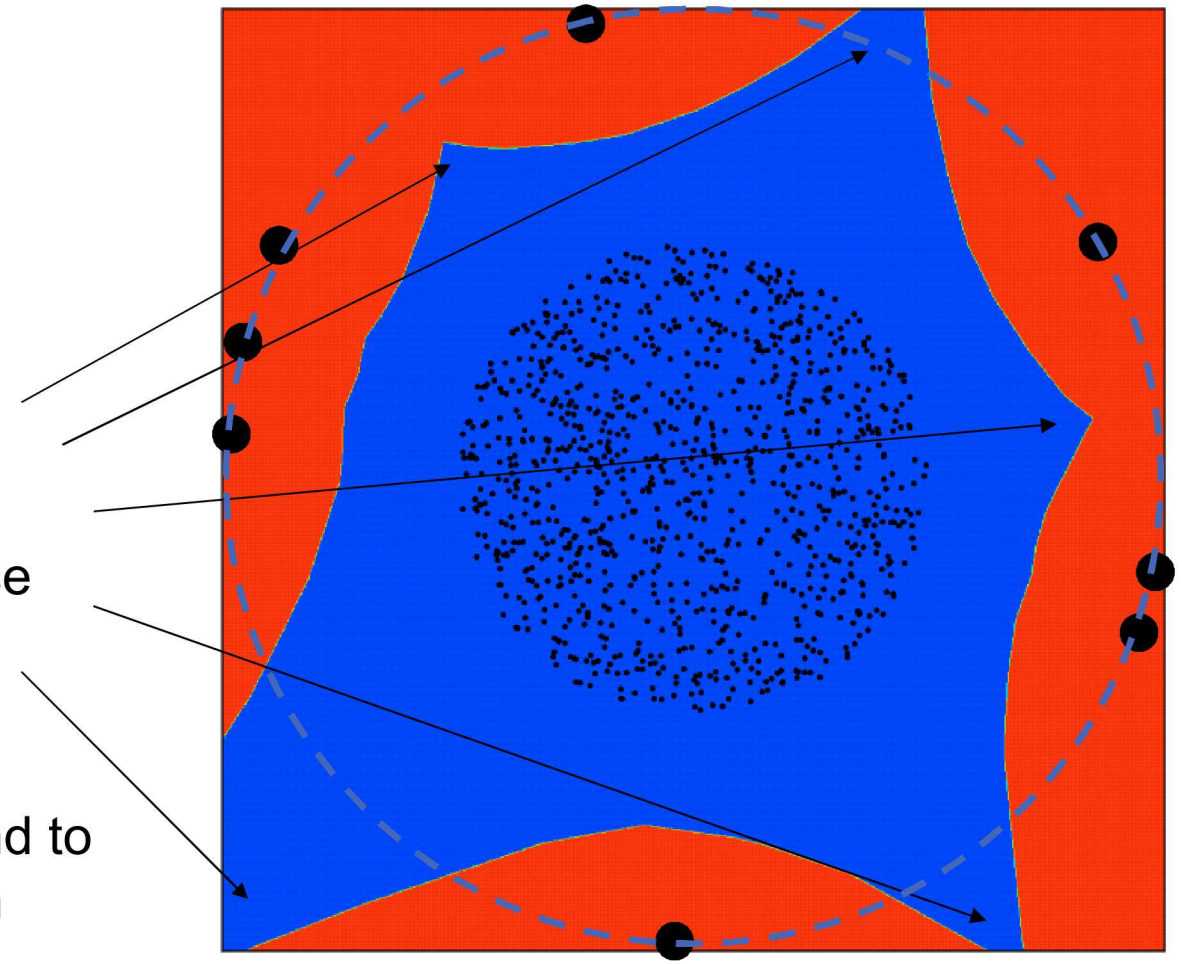
KNN Weighting

- Weighting the points is often used when determining class membership to overcome the “majority voting” problem
  - If one class dominates the data, occurring more frequently, then that class tends to dominate in prediction of a new point because that class membership is most common among the k nearest neighbors.
  - Weighting the classification allows for a fuller representation than ‘’majority rule” in prediction
  - The weight is often proportional to the inverse of the distance (or distance squared) from the new point, x_{new}, to the KNN, $x_i, i = 1..k$.
- The class prediction then involves applying the weights to the classes of the k-nearest neighbors:

$$C(x_{new}) = \sum_{i=1}^{k} w_k \; C(x_k)$$

Relying on distance only can be problematic!!

kNN, k = 1



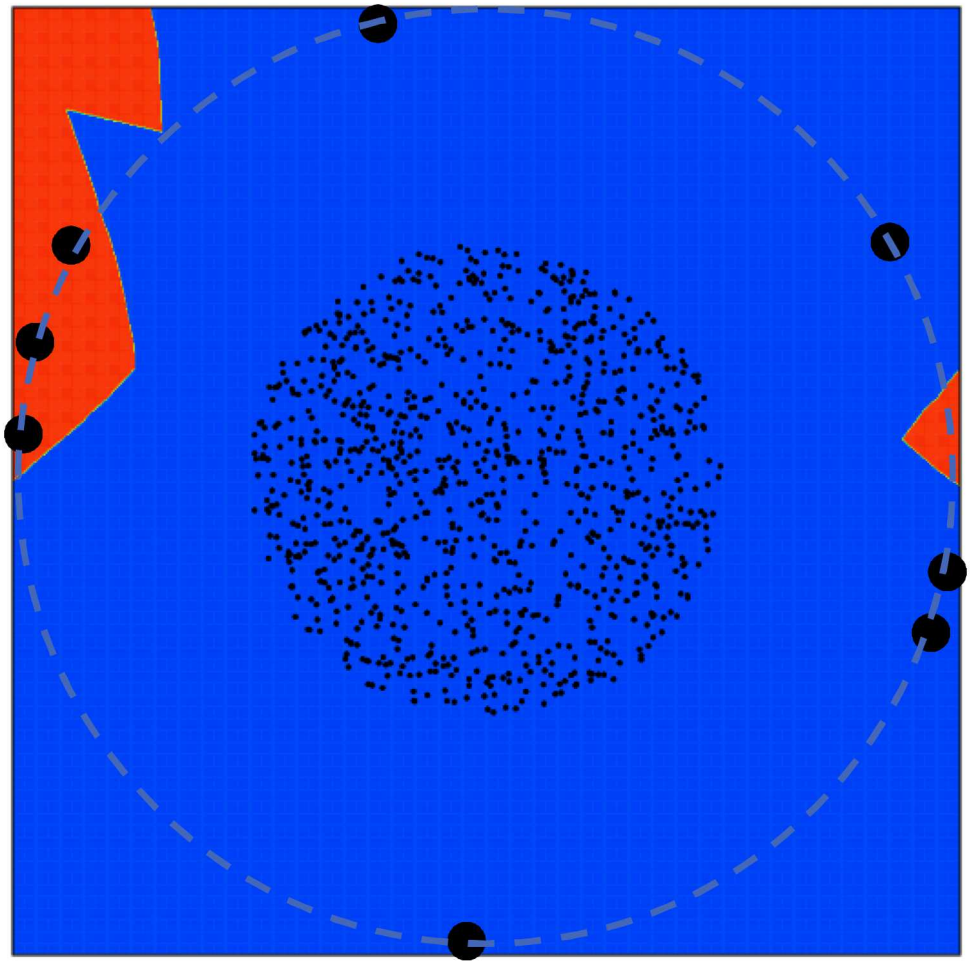
Do you want a classifier with these arms/spikes?

Clustered data tend to propagate through scattered data!!

Relying on distance only can be problematic!!

kNN, k = 5,
no weighting!

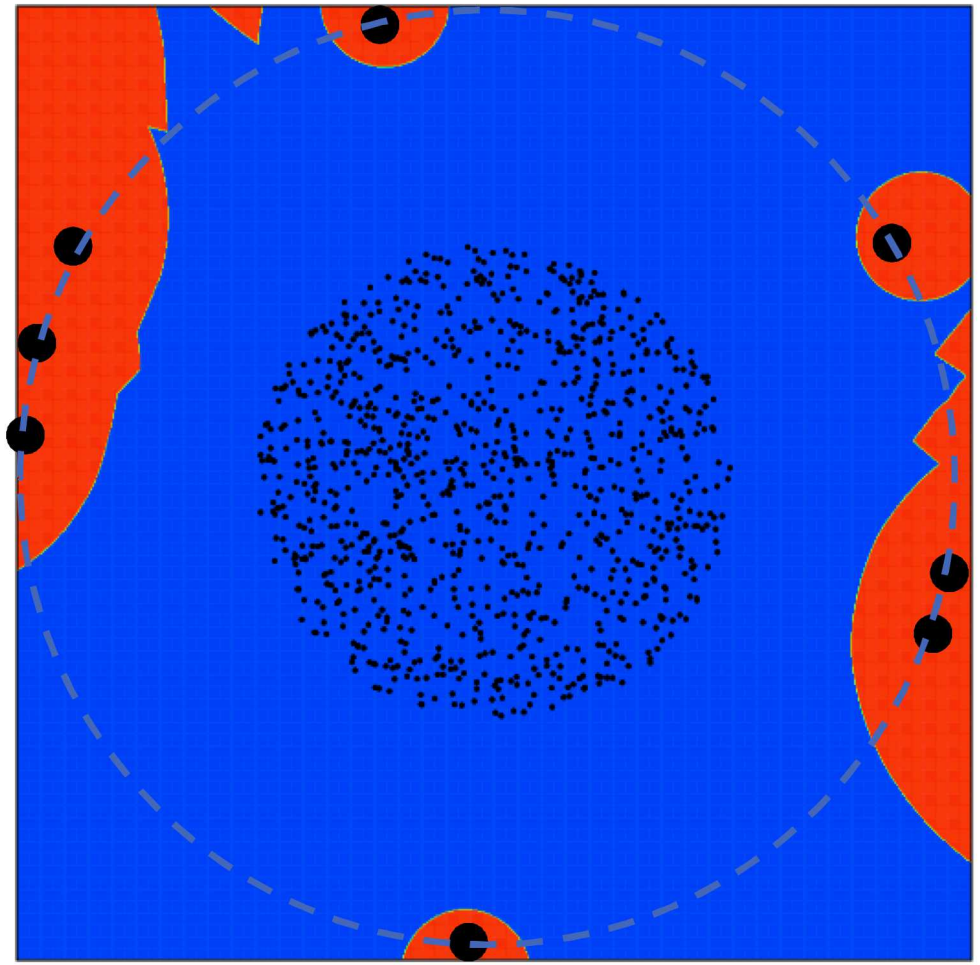
Increasing k without weighting is a TERRIBLE idea!



Relying on distance only can be problematic!!

kNN, k = 5,
+ weighting!

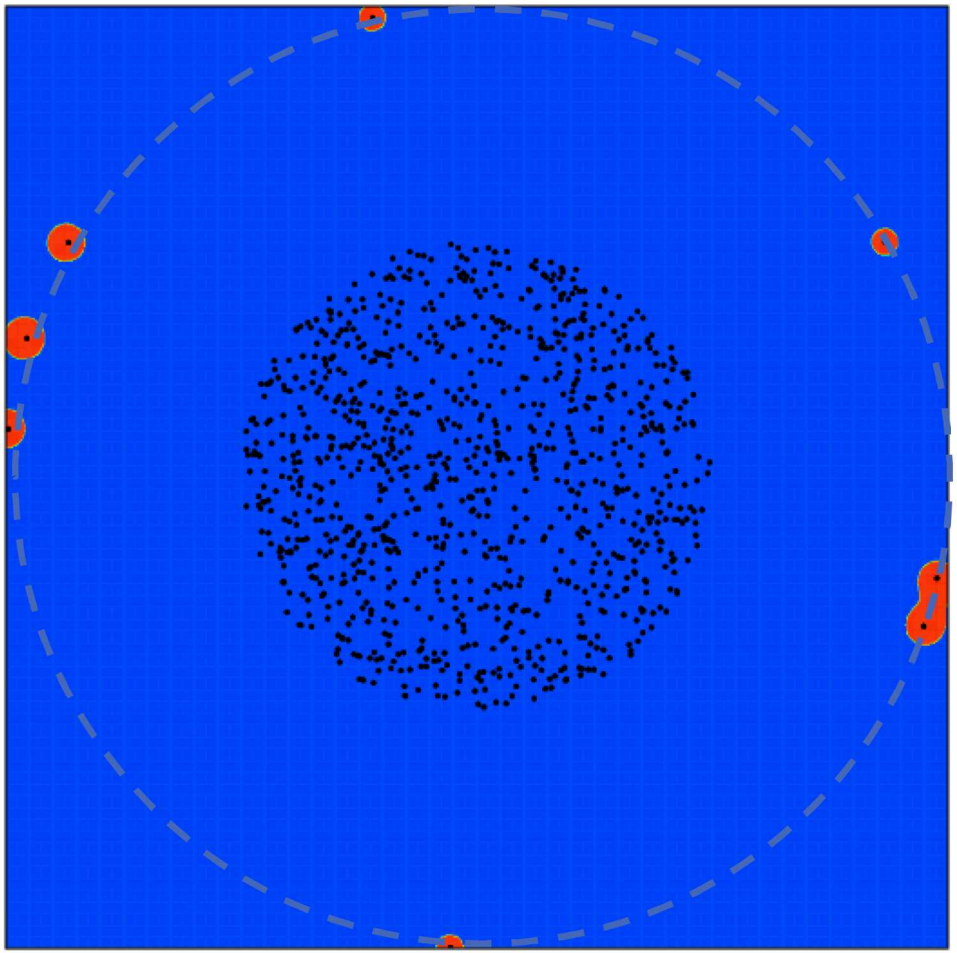
Adding weights improved things a bit yet still not much (still worse than kNN, k = 1!



Relying on distance only can be problematic!!

kNN, k = 20,
+ weighting!

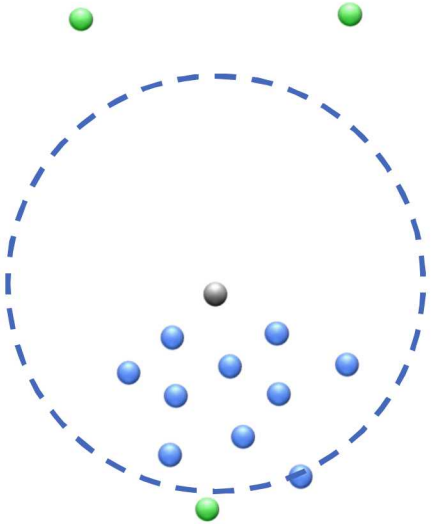
Increasing k makes the situation much WORSE actually!



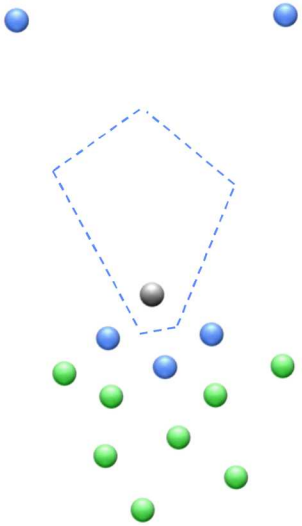
Proposed Approach

Instead of relying on distance only

(nearest neighbors)



, we rely on distance and directions

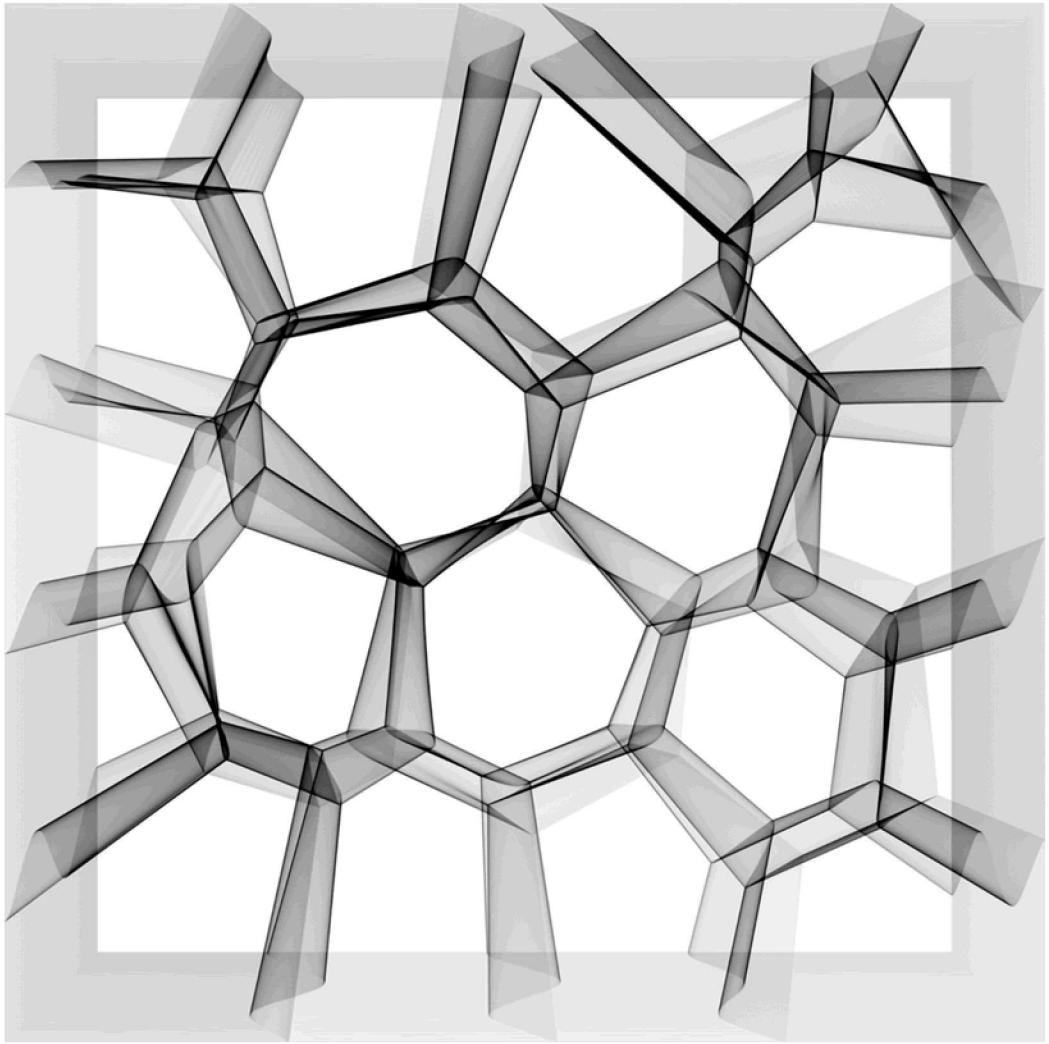


(significant Voronoi neighbors)!

Our Voronoi Solution

→

Our local piecewise classifier partition the space using Voronoi cells.

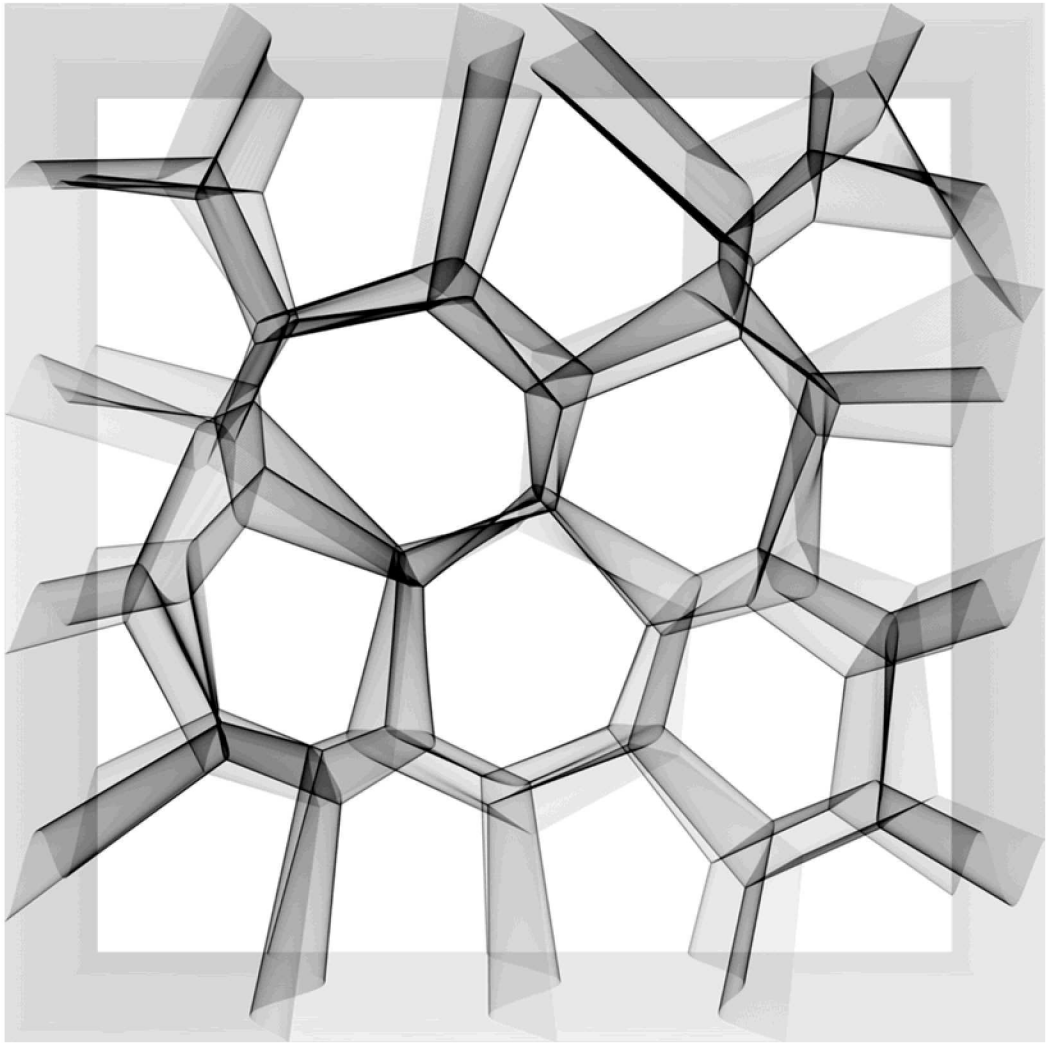


Computational Challenge

Explicit Voronoi Meshing is intractable:

Number of vertices grows exponentially with dimension!

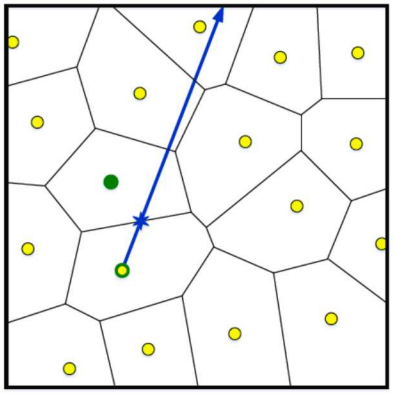
→ [A Curse of dimensionality]



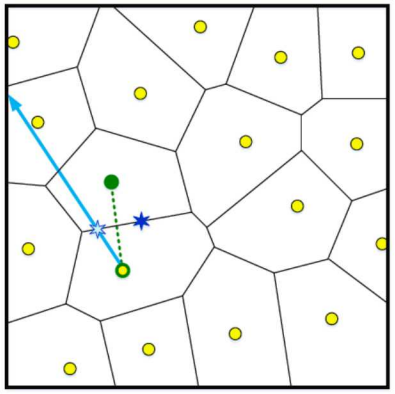
Sandia National Laboratories

Spoke Darts for implicit Voronoi Meshing

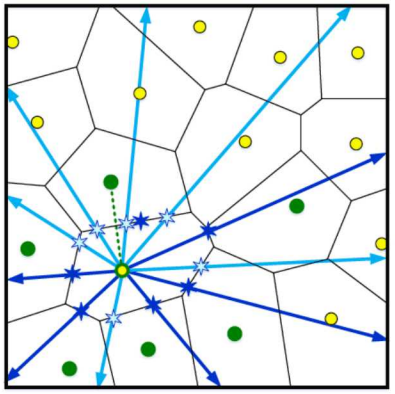
- Spoke-dart: sampling locally from hyper-annuli centered at prior point samples, using lines, planes, or, more generally, hyperplanes.
- The main operation is line-Hyperplane trimming.
- Spoke-dart sampling is tractable in high dimensions yet it is not cheap (each spoke needs to be trimmed by n hyperplanes).



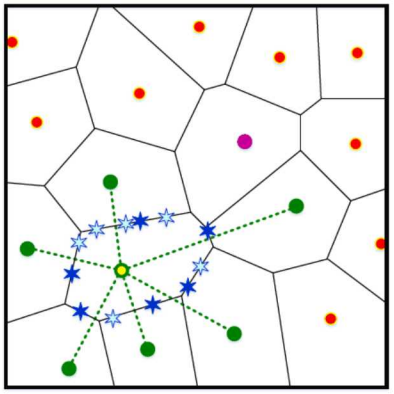
(a) A random line samples a point from a Voronoi facet.



(b) A redundant point sampled from the same facet.

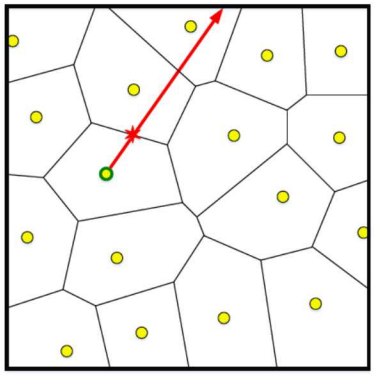


(c) Local MC line sampling to find Delaunay neighbors.

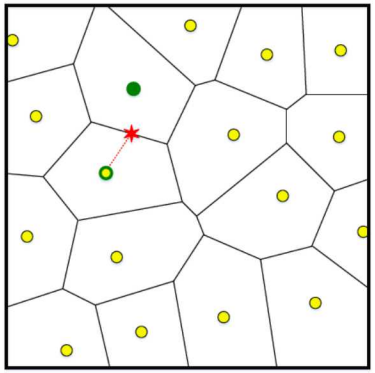


(d) Neighbors sharing relatively small facets missed.

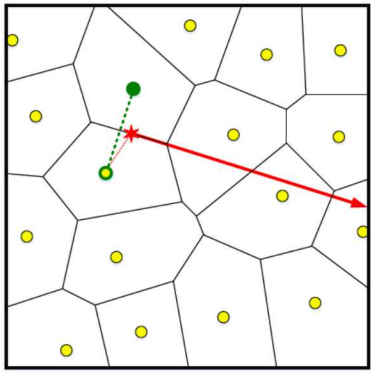
Recursive Spoke darts



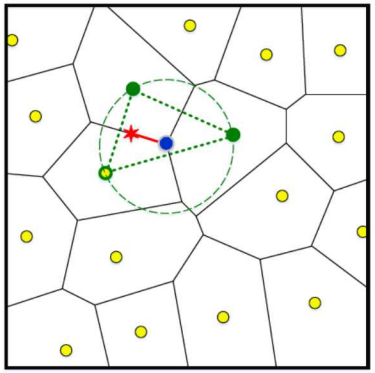
(a) A random ray samples a point from a 2-d space.



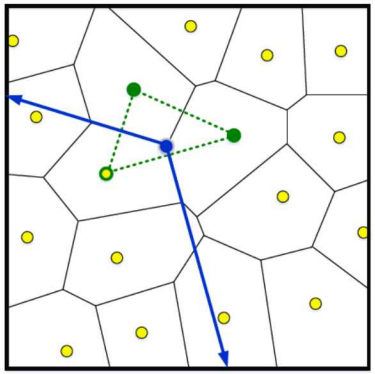
(b) A point is identified as edge and neighbor witness.



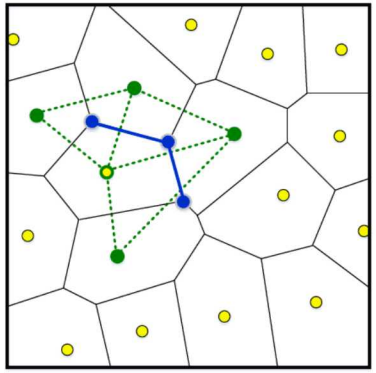
(c) A random ray samples a point from a 1-d edge.



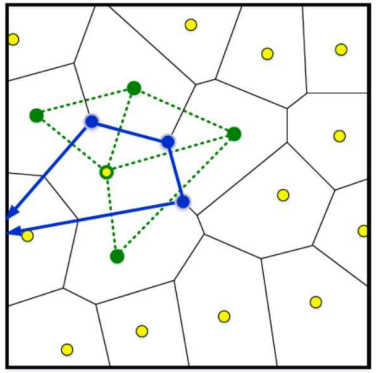
(d) A delaunay simplex around the retrieved vertex.



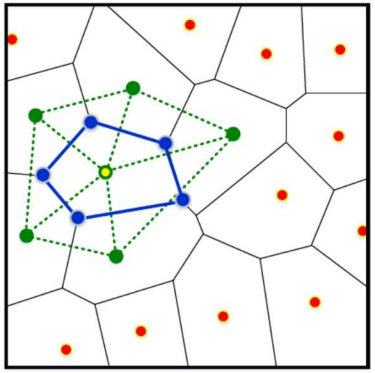
(e) Two line samples from vertex along connected edges.



(f) Neighbor vertices and dual simplices identified.



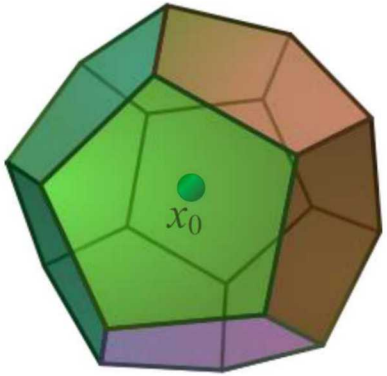
(g) More line samples to identify other verteces.



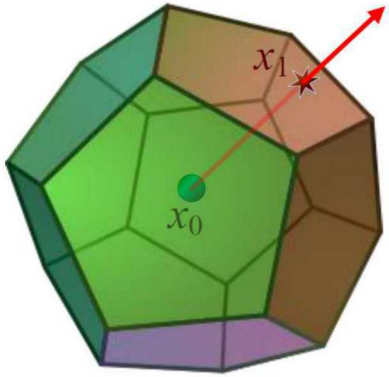
(h) All vertices and their dual simplices identified.

Sandia National Laboratories

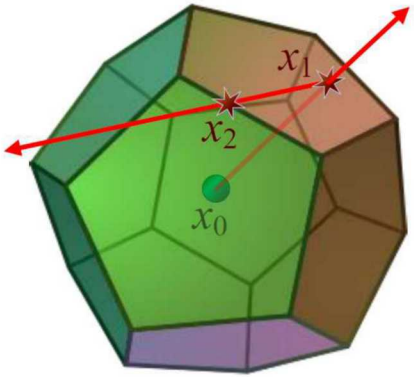
RSD Algorithm in 3-d



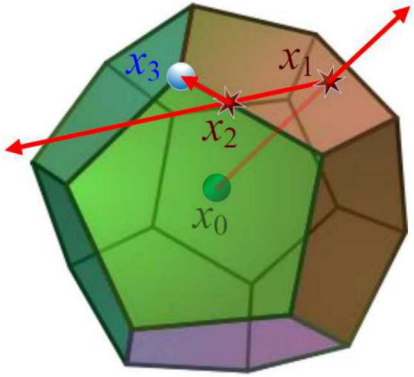
(a) Initial seed selection.



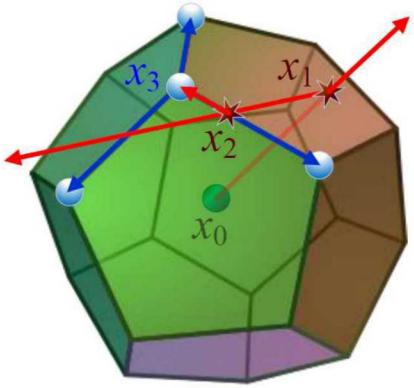
(b) Random 3-d spoke identifies x_1.



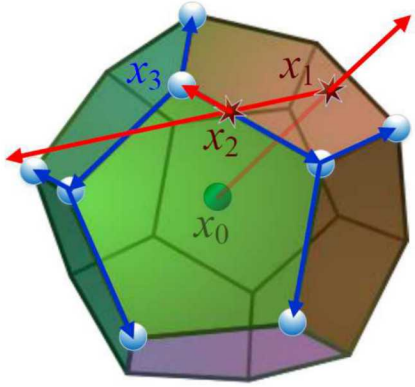
(c) Random 2-d spoke identifies x_2.



(d) Random 1-d spoke identifies x_3.



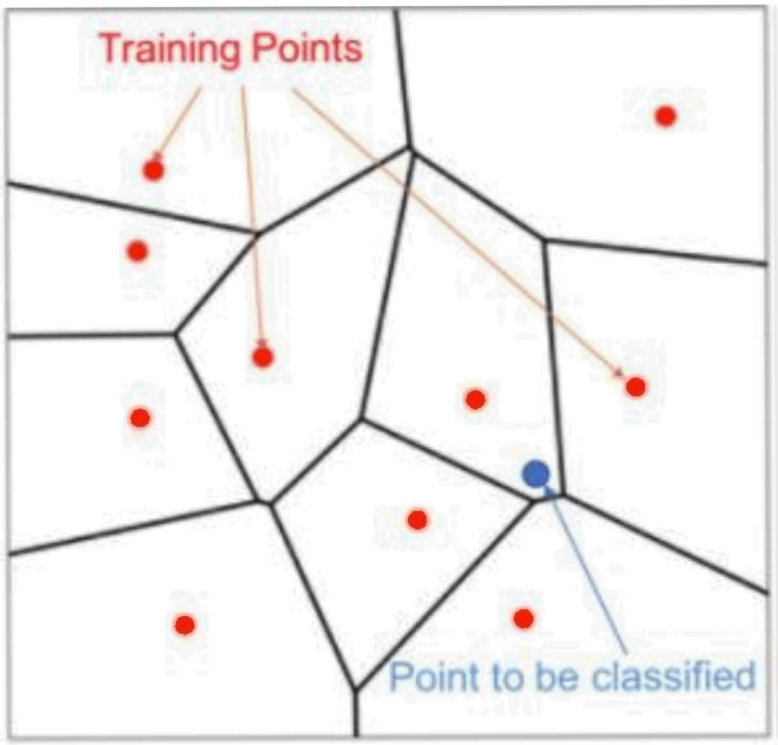
(e) 3 lines identify neighbor vertices.

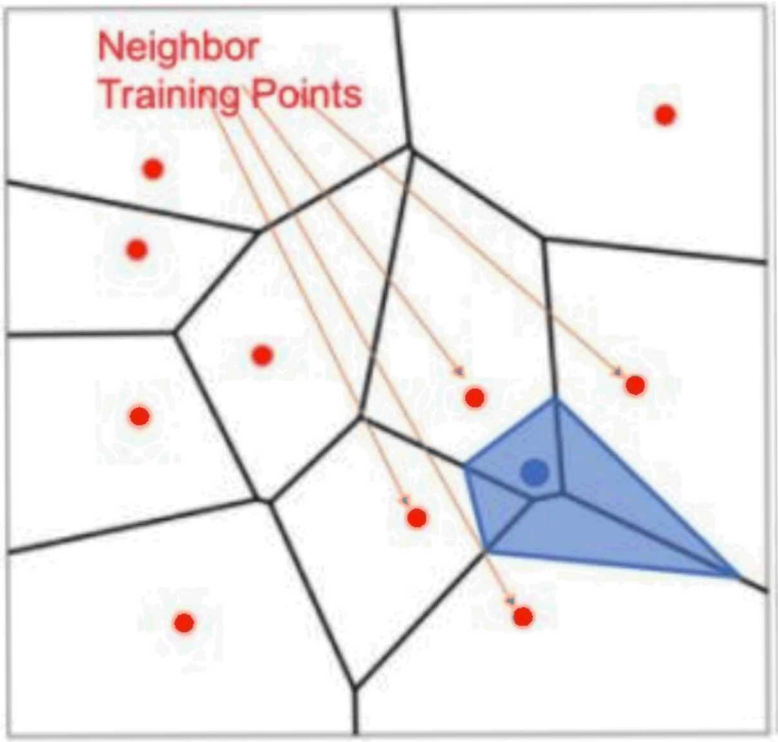


(f) Further line propagation.

Sandia National Laboratories

Our Voronoi Classifier

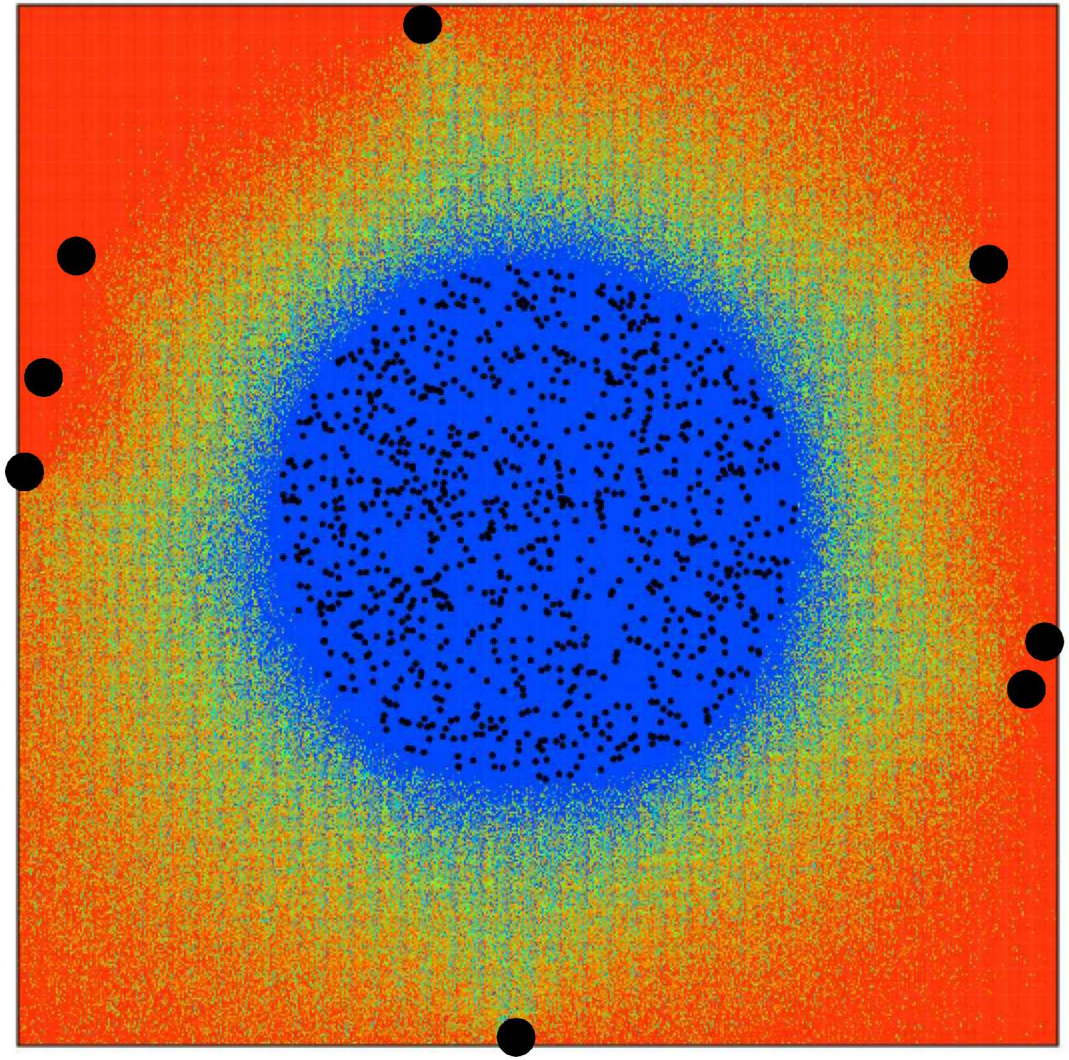




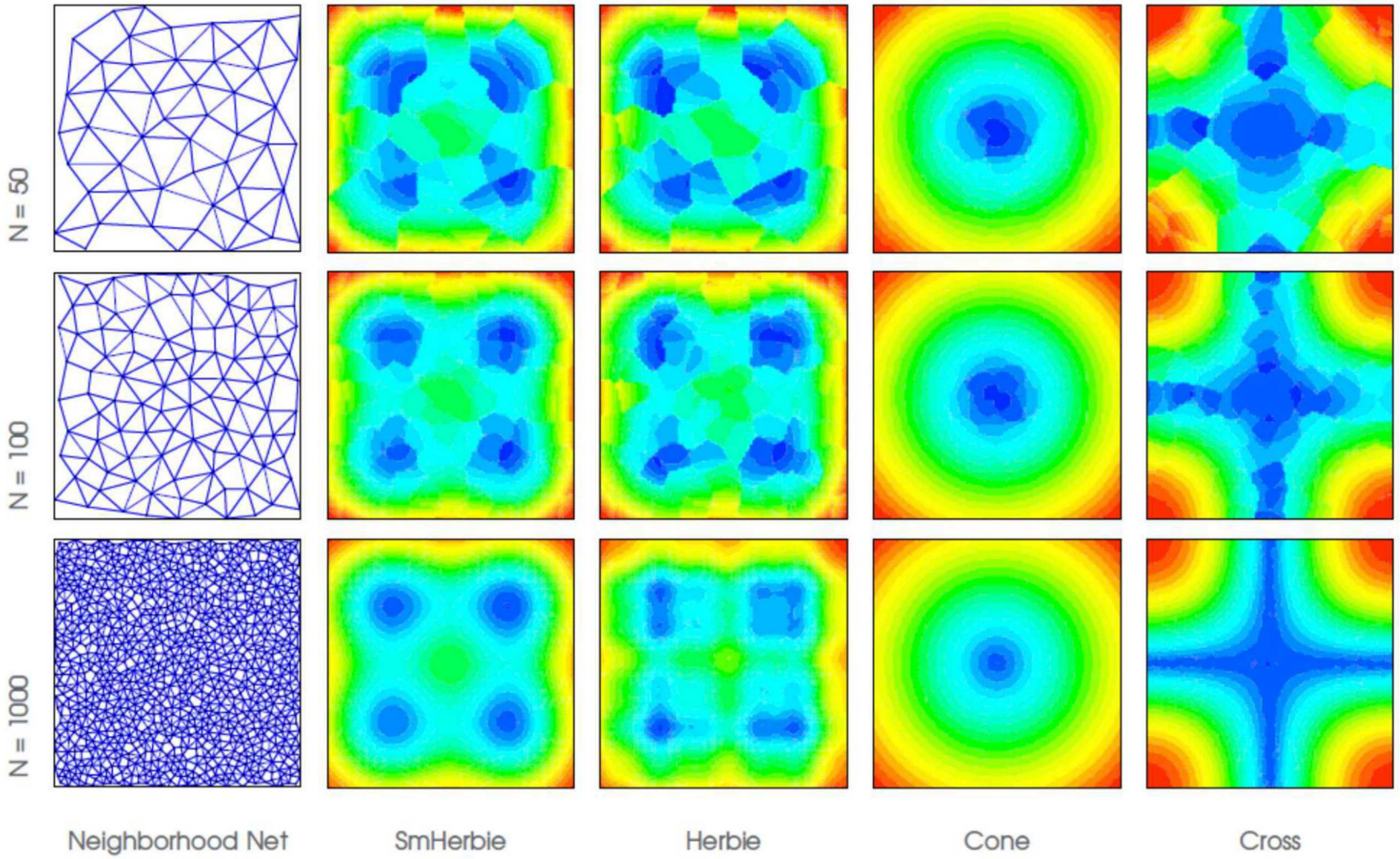
Back to our toy problem, using our Classifier!

10 Spokes
+ weighting!

Our Voronoi classifier limited the undesired propagation of the clustered data

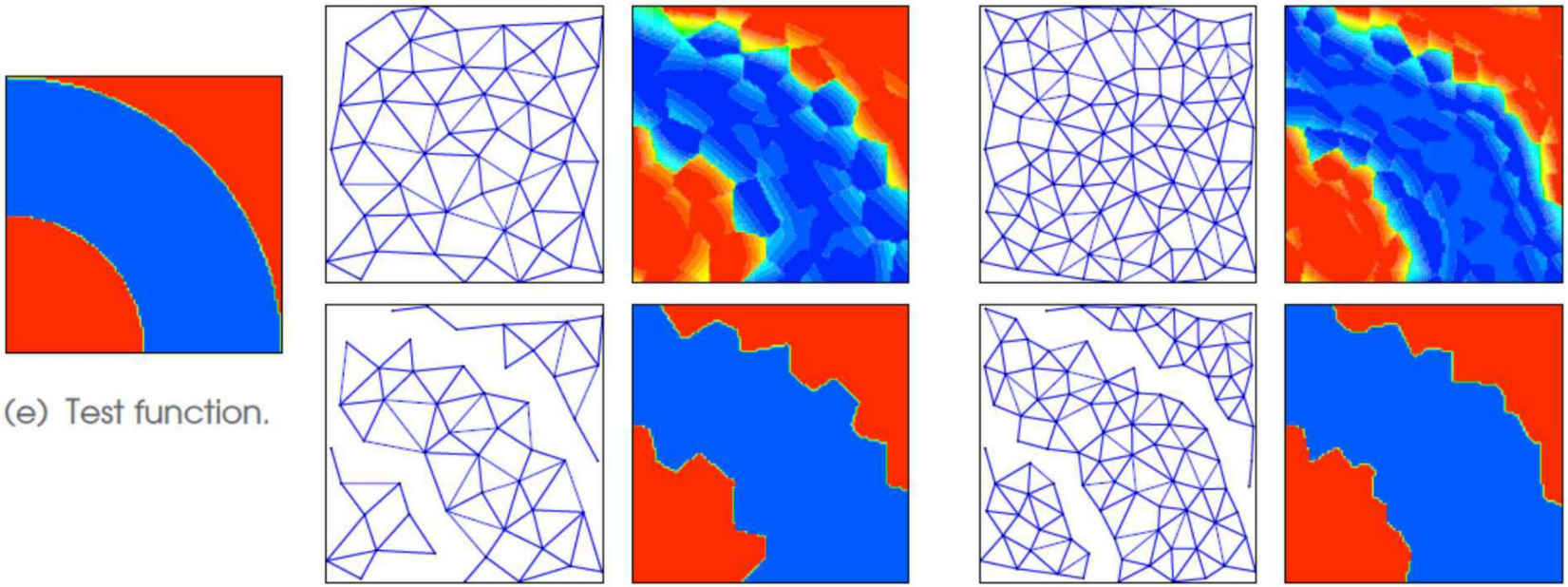


Other application (VPS)



Discontinuity Detection

Breaking Delaunay links across estimated discontinuities.



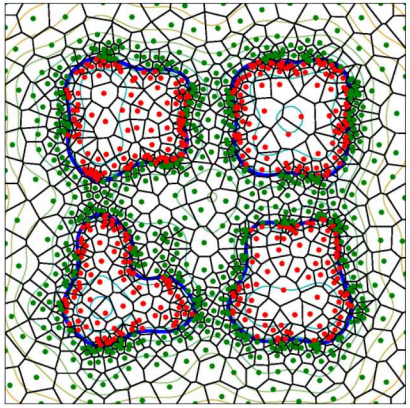
(e) Test function.

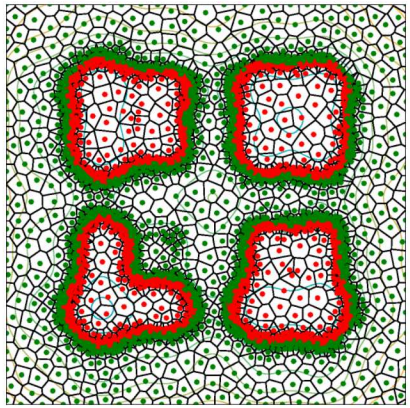
(f) N = 50 evaluations.

(g) N = 100 evaluations.

Probability of failure

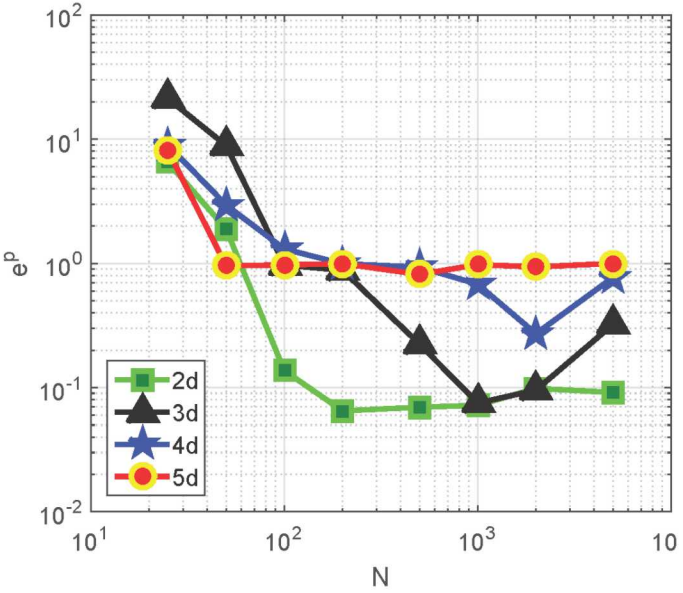
2d version of test function



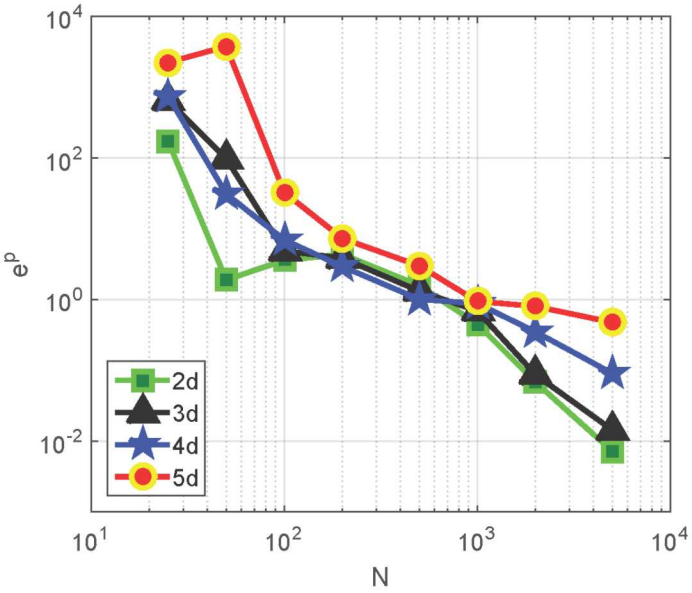


Estimation the probability of failure

Adaptive sampling driven by failure isocontour



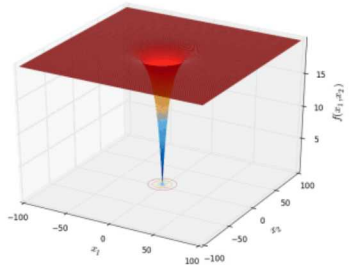
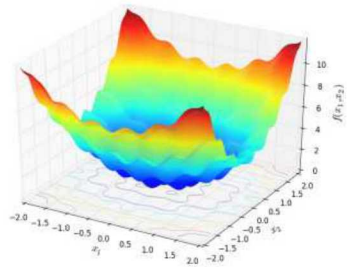
POF-darts + GP



POF-darts + VPS

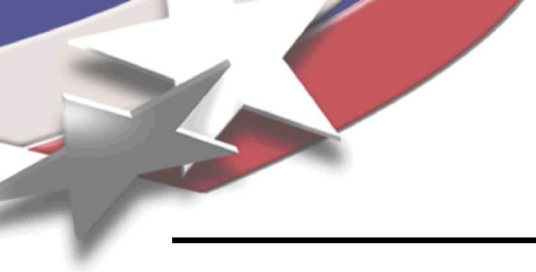
**Ebeida, M. S., Mitchell, S. A., Swiler, L. P., Romero, V. J., & Rushdi, A. A. (2016). Pof-darts: Geometric adaptive sampling for probability of failure. Reliability Engineering & System Safety, 155, 64-77.

Global Optimization

Benchmark f		dimension	DIRECT	Opt-darts	Speedup
Easom		6	5657	1320	4.3 ×
		7	20987	3276	6.4 ×
		8	71677	4814	14.9 ×
		9	257539	14258	18.1 ×
		10	837203	33852	24.7 ×
Bohachevsky		20	5689	1269	4.5 ×
		40	25807	2633	9.8 ×
		60	63765	4345	14.7 ×
		80	122503	6246	19.6 ×
		100	208185	7802	26.7 ×

Ebeida, Mohamed S., et al. ”Spoke darts for efficient high dimensional blue noise sampling ." Arxiv, TOG *accepted* (2018): 110-122.

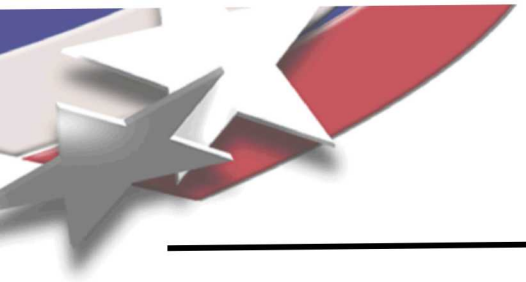
Sandia National Laboratories



Thank You!

Acknowledgment: This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR) Applied Mathematics Program, and the NNSA Advanced Scientific Computing (ASC) Program. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories



Backup slides

Use of Spoke-Darts

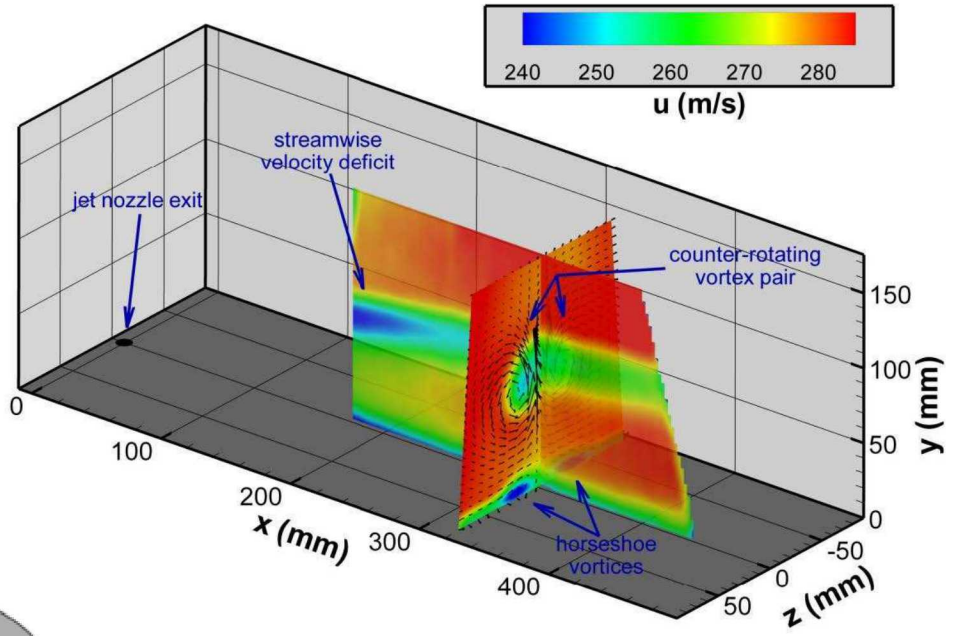
- Ideas from computational geometry
  - a bunch of “spokes” are thrown that go through an existing sample point. The spokes are then “trimmed” so that the end of the spokes can be used to find approximate Voronoi vertices surrounding an existing sample point and efficiently find neighbors which are centroids of adjoining Voronoi cells.
- Spoke-dart performance better when the nearest neighbors to a new testing point in feature space are clustered on “one side” of the point.
  - The standard algorithm would select only the close neighbors all in one region or area.
  - With spoke-darts, neighbors are gathered from all directions regardless of their distance if the Voronoi cells for a training point x_{train} and x_{new} intersect.
  - The number of neighbors is determined by the number of Voronoi neighbors of x_{new}. No *k* is specified in this case. This is an advantage because is it not clear how to choose *k* a priori, and optimization methods to determine the best *k* are computationally expensive.

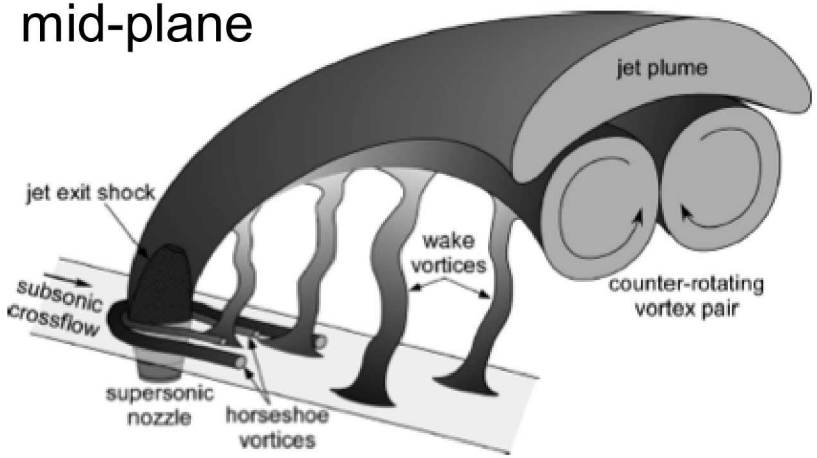
Spoke-Dart Classification in UQ

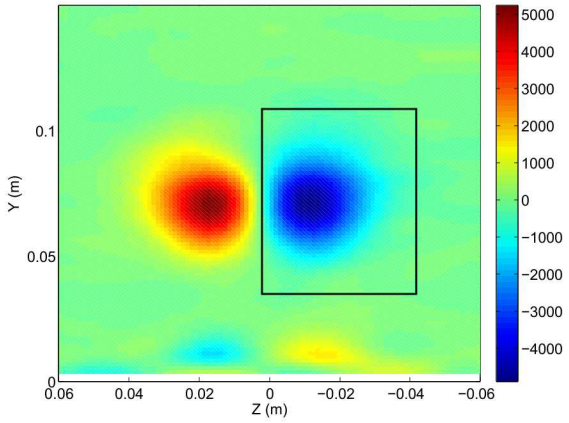
- We can use this Voronoi-based classification approach for UQ
- If points in the feature space are labeled 'infeasible' or unphysical, we can estimate the size based on the sum of the area of the infeasible Voronoi cells. This quantity can be used as an estimate of the probability of failure or infeasibility.
- The classification can define the isocontours between failure and safe or feasible regions.
- We will build on previous work that used Voronoi cells to estimate probability of failure [Ebeida et al. 2016]. In the case where the "feasible" classification also has numerical quantities of interest, we can use the Voronoi cells to construct local surrogate models for prediction where we "automatically" get the infeasible points removed from consideration. This is a big advantage in many computational simulations.

Target problem - jet-in-crossflow

- A canonical problem for spin-rocket maneuvering, fuel-air mixing etc.
- We have experimental data (PIV measurements) on the cross- and mid-plane
- Will calibrate to vorticity on the crossplane and test against mid-plane

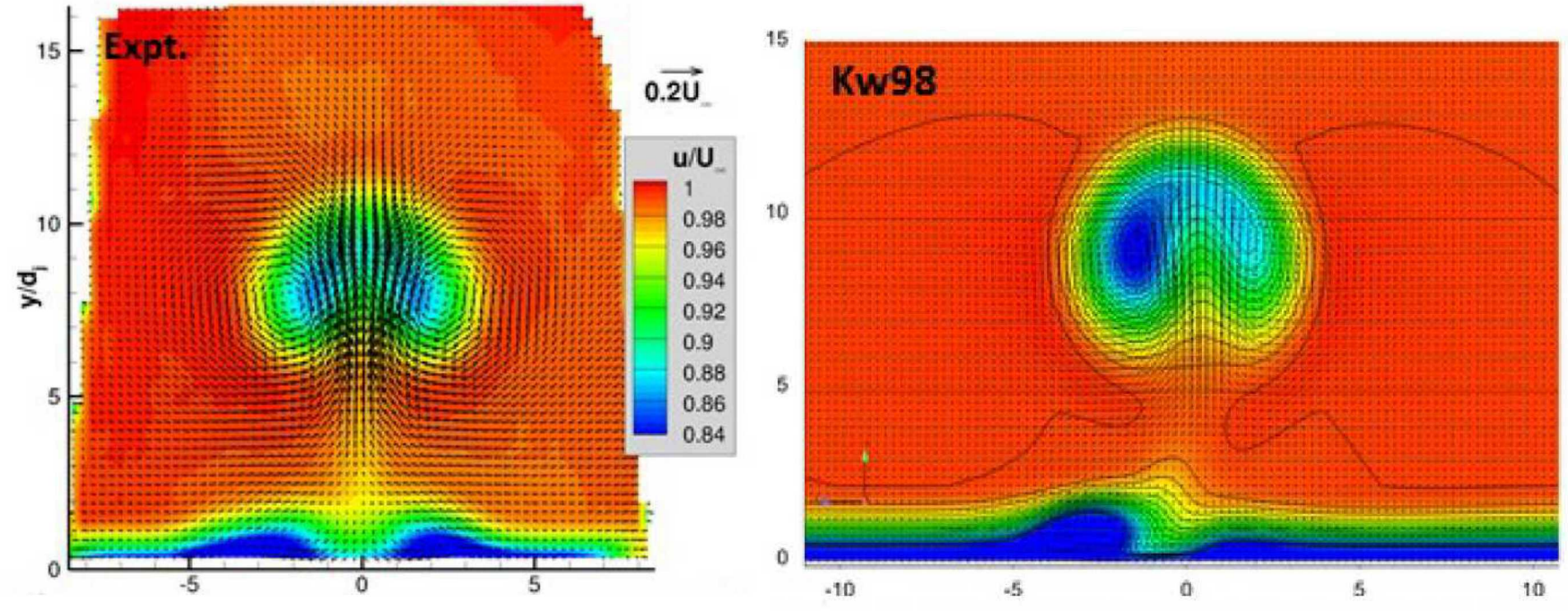






Sandia National Laboratories

RANS (k-ω) simulations - crossplane results



- Crossplane results for stream
- Computational results (SST) are too round; Kw98 doesn't have the mushroom shape; non-symmetric!
- Less intense regions; boundary layer too weak

- The model
  - Devising a method to calibrate 3 k-ε parameters **C** = {C_μ, C_2, C_1} from expt. data

$$\frac{\partial \rho k}{\partial t} + \frac{\partial}{\partial x_i}\left[\rho u_i k - \left(\mu + \frac{\mu_T}{\sigma_k}\right)\frac{\partial k}{\partial x_i}\right] = P_k - \rho\varepsilon + S_k$$

$$\frac{\partial \rho \varepsilon}{\partial t} + \frac{\partial}{\partial x_i}\left[\rho u_i \varepsilon - \left(\mu + \frac{\mu_T}{\sigma_\varepsilon}\right)\frac{\partial \varepsilon}{\partial x_i}\right] = \frac{\varepsilon}{k}\left(C_1 f_1 P_k - C_2 f_2 \rho\varepsilon\right) + S_\varepsilon$$

$$\mu_T = C_\mu f_\mu \rho \frac{k^2}{\varepsilon}$$

- Calibration parameters
  - **C** = {C_μ, C_2, C_1} ; C_μ: affects turbulent viscosity; C_1 & C_2: affects dissipation of TKE
- Calibration method
  - Pose a statistical inverse problem using experimental data
  - Estimate parameters using Markov chain Monte Carlo (MCMC)
- Once calibrated, address the problem of approximated physics (model-form error)

Making emulators - 1

- Training data
  - Parameter space C: $0.06 < C\mu < 0.12$; $1.7 < C_2 < 2.1$; $1.2 < C_1 < 1.7$
  - $\mathbf{C}_{nom} = \{0.09, 1.93, 1.43\}$
  - Take 2744 samples in C using a space-filling quasi Monte Carlo pattern
    - Save the streamwise vorticity field $\omega_x(\mathbf{y}; \mathbf{C})$
- Choosing the "probes"
  - Will try to create emulators for each grid cell on the crossplane
  - Most grid cells have lots of numerical noise
  - For a given run, choose the grid cells with vorticity the top 25% percentile (56 grid cells)
  - Take the union of such grid cells, union over the 2744 members of the training set (comes to 108 grid cells)
    - We will try to make emulators for these 108 grid cells with large vorticity

Making emulators - 2

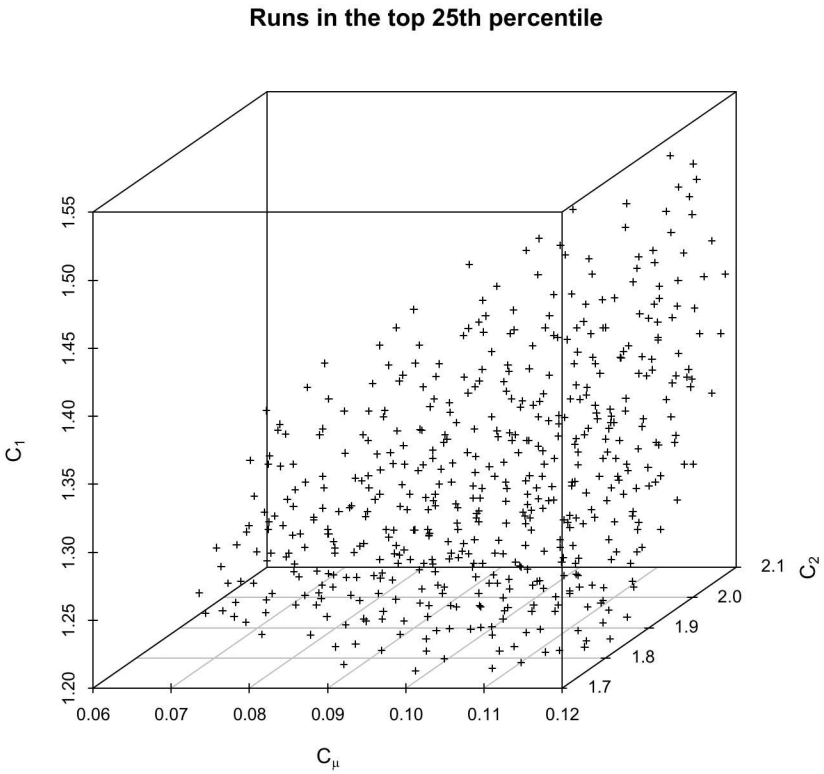
- Model ω_x in grid cell *j* as a function of **C** i.e. $\omega^{(j)}_x = f^{(j)}(\mathbf{C})$
  - Approximate this dependence with a polynomial

$$\omega^{(j)} \cong a_0 + a_1 C_\mu + a_2 C_2 + a_3 C_1 + a_4 C_\mu C_2 + a_5 C_\mu C_1 + a_6 C_2 C_1 + ...$$

- But how to get (a_0, a_1, ….) for each of the probe locations to complete the surrogate model for each probe?
  - Divide training data in a Learning Set and Testing Set
  - Fit a full cubic model for to the Learning Set via least-squares regression; sparsify using AIC
  - Estimate prediction RMSE for Learning & Testing sets; should be equal
- Final model tested using 100 rounds of cross-validation
- 10% error threshold was used to select models for the probes

Making emulators - 3

- Choosing $\mathcal{R}$
  - emulators failed – we could not model any surrogates to within 10% accuracy
  - This is because many **C** = {Cμ, C_2, C_1} combination are nonphysical
  - We compute the RMSE vorticity difference between the training set RANS runs and experimental observations
    - We retain only the top 25 percentile of the runs (using RMSE) as training data ($\mathcal{R}$)

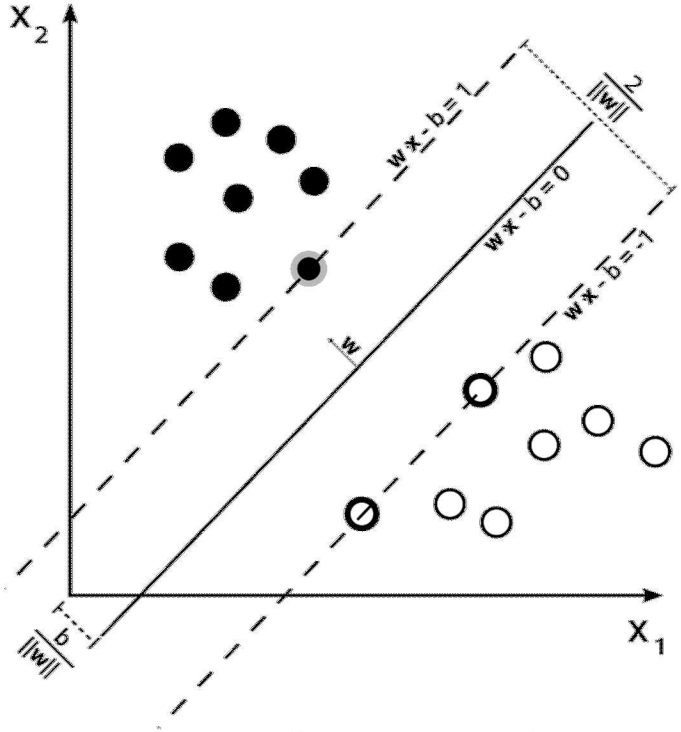


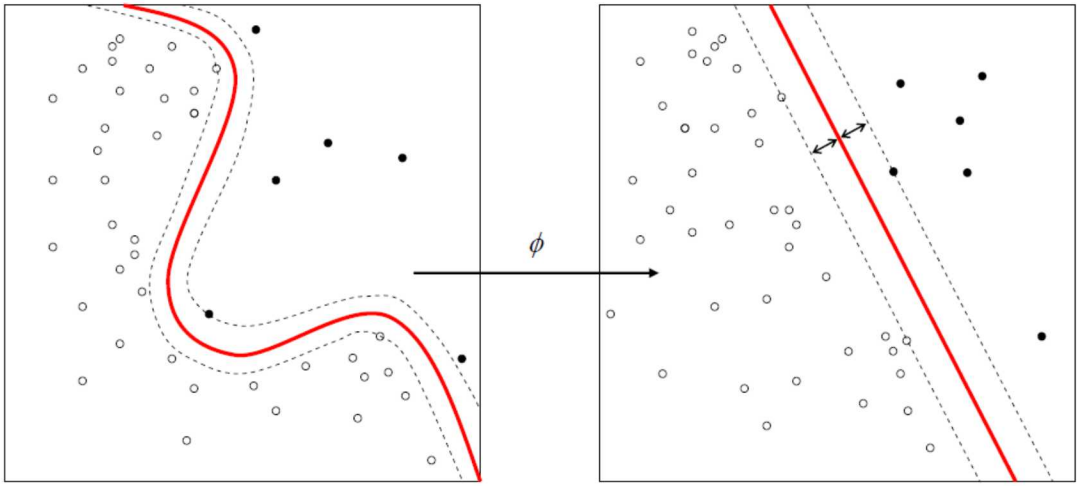
Making the informative prior

- Our emulators are valid only inside $\mathcal{R}$ in the parameter space C
- During the optimization (MCMC) we have to reject parameter combinations outside $\mathcal{R}$ (this is our prior belief $\pi_{prior}(\mathbf{C})$)
  - We define $\zeta(\mathbf{C}) = 1$, for **C** in R and $\zeta(\mathbf{C}) = -1$ for **C** outside $\mathcal{R}$
  - Then the level set $\zeta(\mathbf{C}) = 0$ is the boundary of $\mathcal{R}$
- The training set of RANS runs is used to populate $\zeta(\mathbf{C})$
- We have to "learn" the discriminating function $\zeta(\mathbf{C}) = 0$
  - We'll do that using support vector machine (SVM) classifiers

What is a SVM classifier?

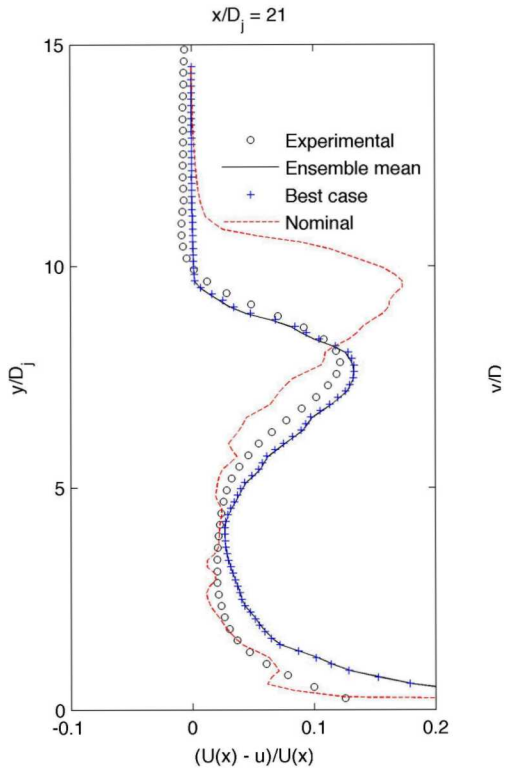
- Given a binary function y = f(**x**) as a set of points (y_i, $\mathbf{x}_i$), y_i = (0, 1)
  - Find the hyperplane y + Ax = 0 that separates the x-space into y = 0 and y = 1 parts
- Posed as an optimization problem that maximizes the margin

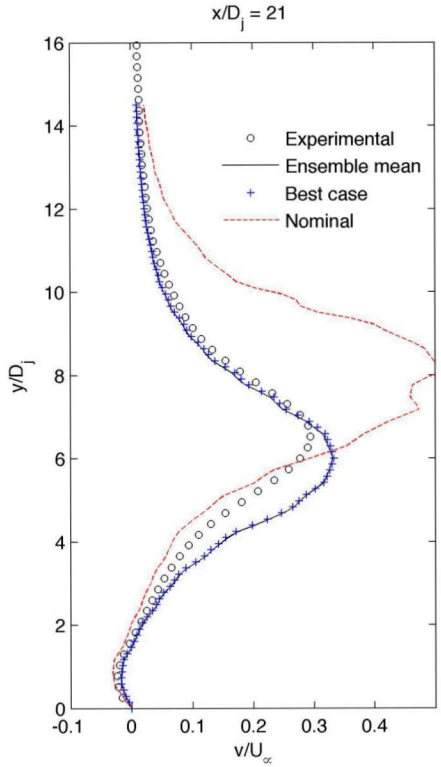




- In case of a curved discriminator, need a transformation first
  - Achieved using kernels
  - We use a cubic kernel

Check # 3 – mid-plane comparisons





Streamwise velocity deficit at x/D = 21

Vertical velocity at x/D = 21

- Flow quantities on the mid-plane were not used in the calibration