Stable Sparse Interpolation with Fewer Samples

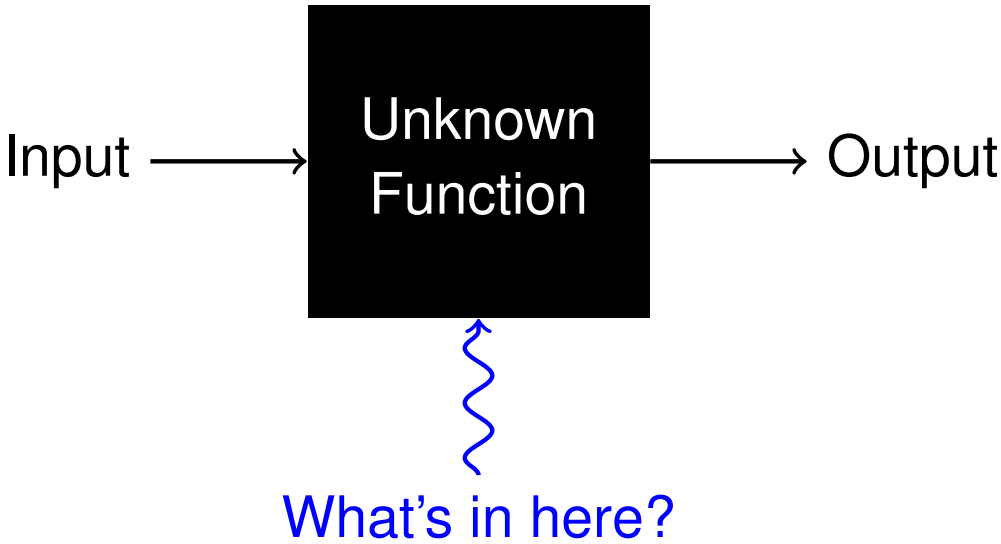
Daniel S. Roche
United States Naval Academy

joint work with
Mark Giesbrecht
University of Waterloo

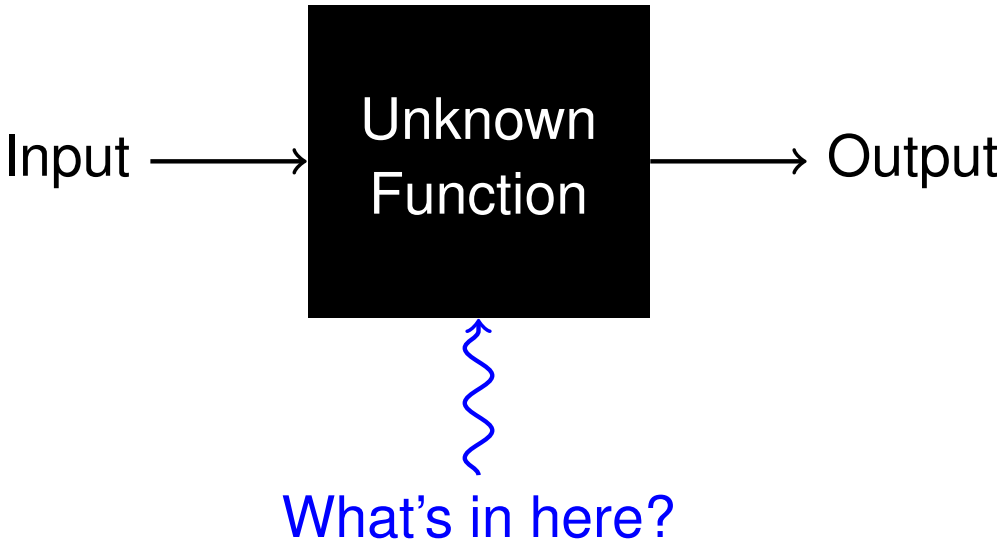
Fields Workshop on Hybrid Methodologies for Symbolic-Numeric Computation

November 18, 2011

The Problem



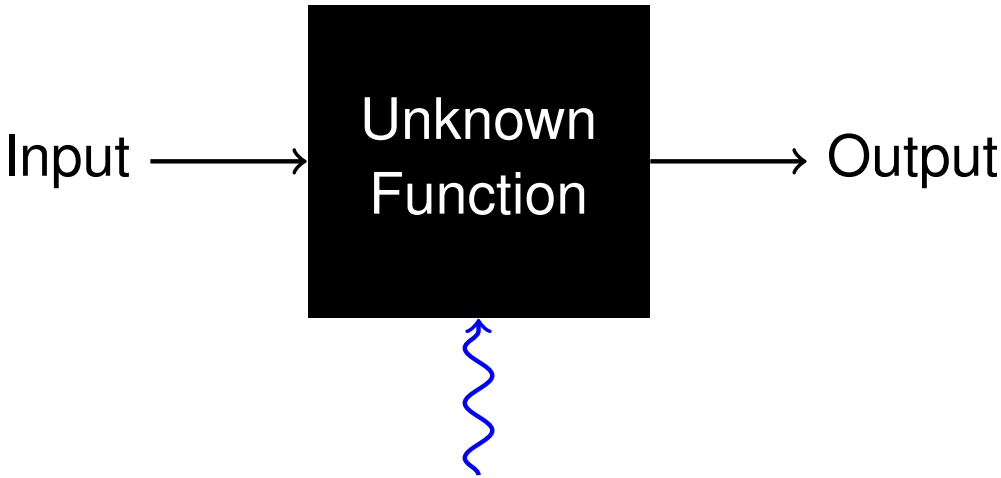
The Problem



Algorithm input: The black box

Algorithm output: The unknown function

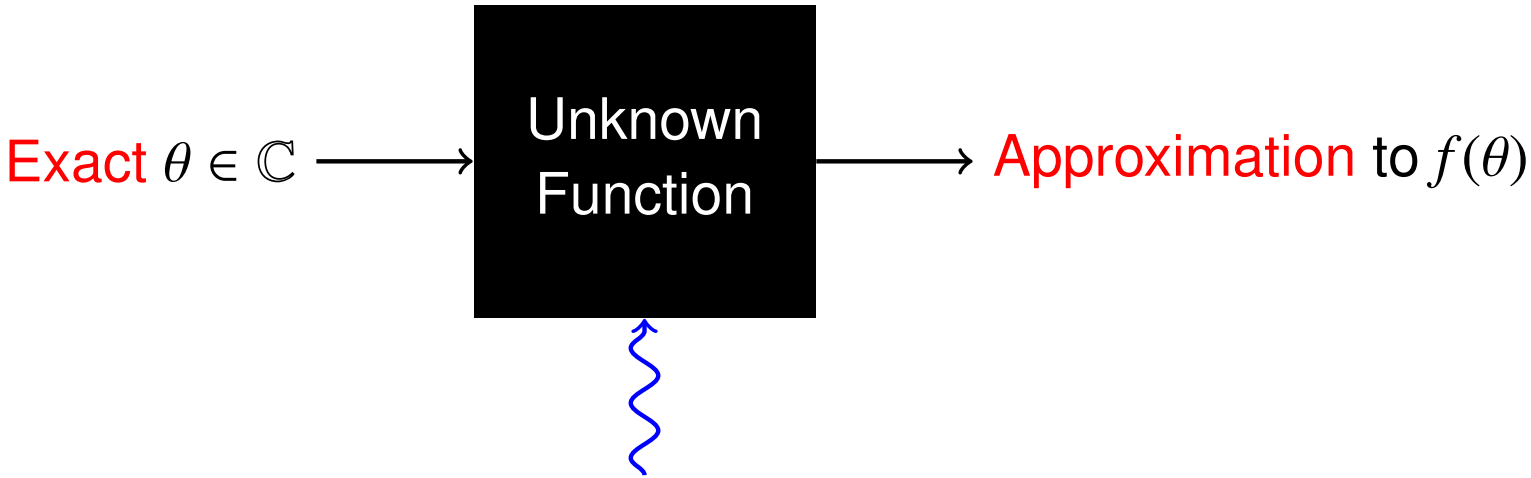
The Problem



Algorithm input: Black box for a sparse polynomial

Algorithm output: Approximation to the hidden polynomial

The Problem



Algorithm input: Way to evaluate $f(x) = \sum_{1 \le i \le t} a_i x^{d_i}$
Bounds $D \ge d_i$ and $T \ge t$

Algorithm output: Exponents $d_1, \ldots, d_t$
Coefficients $c_1, \ldots, c_t$

Problem is Inherently Symbolic-Numeric!

	Exact	Approximate
Input	Bounds	Evaluations
Output	Exponents	Coefficients

Factors Influencing Complexity

- **Sparsity** (number of nonzero terms)
- **Degree** (largest exponent)
- **Precision** (error in evaluations)

Our interest is in the hardest case:
High sparsity, high degree, low precision

We want to minimize the number of probes
and the post-processing cost.

de Prony's Method

Algorithm to interpolate exponential sums.
Involves structured linear system solving, polynomial root finding, and computing logarithms.

Much attention in recent years:

- Ben-Or & Tiwari ('88)
- Kaltofen & Lakshman ('89)
- Kaltofen, Lakshman, Wiley ('90)
- Kaltofen, Yang, Zhi ('07)
- Cuyt & Lee ('08)
- Giesbrecht, Labahn, Lee ('09)

Properties of de Prony's method

Drawbacks

- Not numerically stable
- Requires high precision
- (Discrete logarithms?)

Advantages

- Fewest number of evaluations: $O(t)$
- Numerical stability can be helped with randomization

Degree Reduction

Basic Idea: Given a sparse polynomial's black box,
choose evaluations carefully to simulate a lower-degree polynomial

Typically, we get $f \bmod (x^p - 1)$ or $f \bmod (x^{p-1} - 1)$.

Some appearances:

- Bläser, Hardt, Lipton, Vishnoi ('09)
- Garg & Schost ('09)
- G. & R. ('10)

Garg & Schost's Algorithm

Consider (unknown) $f = c_1 x^{e_1} + c_2 x^{e_2} + \cdots + c_t x^{e_t}$.

Idea: Evaluate f mod $x^p - 1$ for a small prime p.
This gives $f_p = c_1 x^{e_1 \bmod p} + c_2 x^{e_2 \bmod p} + \cdots + c_t x^{e_t \bmod p}$.

If p is "good", then every $e_i \bmod p$ is distinct, and we have every coefficient and an unordered set $\{e_i \bmod p \mid 1 \leq i \leq t\}$.

Problem: How to correlate terms between different evaluations?

Garg & Schost's Algorithm

Consider (unknown) $f = c_1 x^{e_1} + c_2 x^{e_2} + \cdots + c_t x^{e_t}$.

Idea: Evaluate f mod $x^p - 1$ for a small prime p.
This gives $f_p = c_1 x^{e_1 \bmod p} + c_2 x^{e_2 \bmod p} + \cdots + c_t x^{e_t \bmod p}$.

If p is "good", then every $e_i \bmod p$ is distinct, and we have every coefficient and an unordered set $\{e_i \bmod p \mid 1 \leq i \leq t\}$.

Problem: How to correlate terms between different evaluations?

Consider the symmetric polynomial whose roots are the exponents: $\Gamma(z) = (z - e_1)(z - e_2) \cdots (z - e_t) \in \mathbb{Z}[z]$.

Coefficients of Γ have $\Theta(t \log d)$ bits, so we need this many "good prime" evaluations. Then we must find the integer roots of Γ.

Making Garg & Schost Numeric

The previous algorithm was for finite fields.

For other domains, we need a way to compute $f \bmod (x^p - 1)$ for a chosen p.

This is easy in $\mathbb{C}$: Evaluate $f(1), f(\omega), \ldots, f(\omega^{p-1})$ for ω a p-PRU. Using the FFT, this is perfectly numerically stable!

Essentially, we are oversampling to get the best stability.

Diversification

Goal: Avoid the need for computing the symmetric polynomial from Garg & Schost.

- Define *diverse polynomial* as one with pairwise-distinct coefficients.
- If α is a random element (of a certain domain), $f(\alpha x)$ is diverse w.h.p.
- We can of course recover $f(x)$ from $f(\alpha x)$.

Result: Cost (number of probes and post-processing) reduced from $O(t^4 \log^2 d)$ down to $O(t^2 \log^2 d)$.

This is work in progress!

Improving on Diversification

$O(t^2 \log^2 d)$ is an improvement from Garg & Schost,
but quadratically more probes than de Prony.

New idea — Use the old idea!

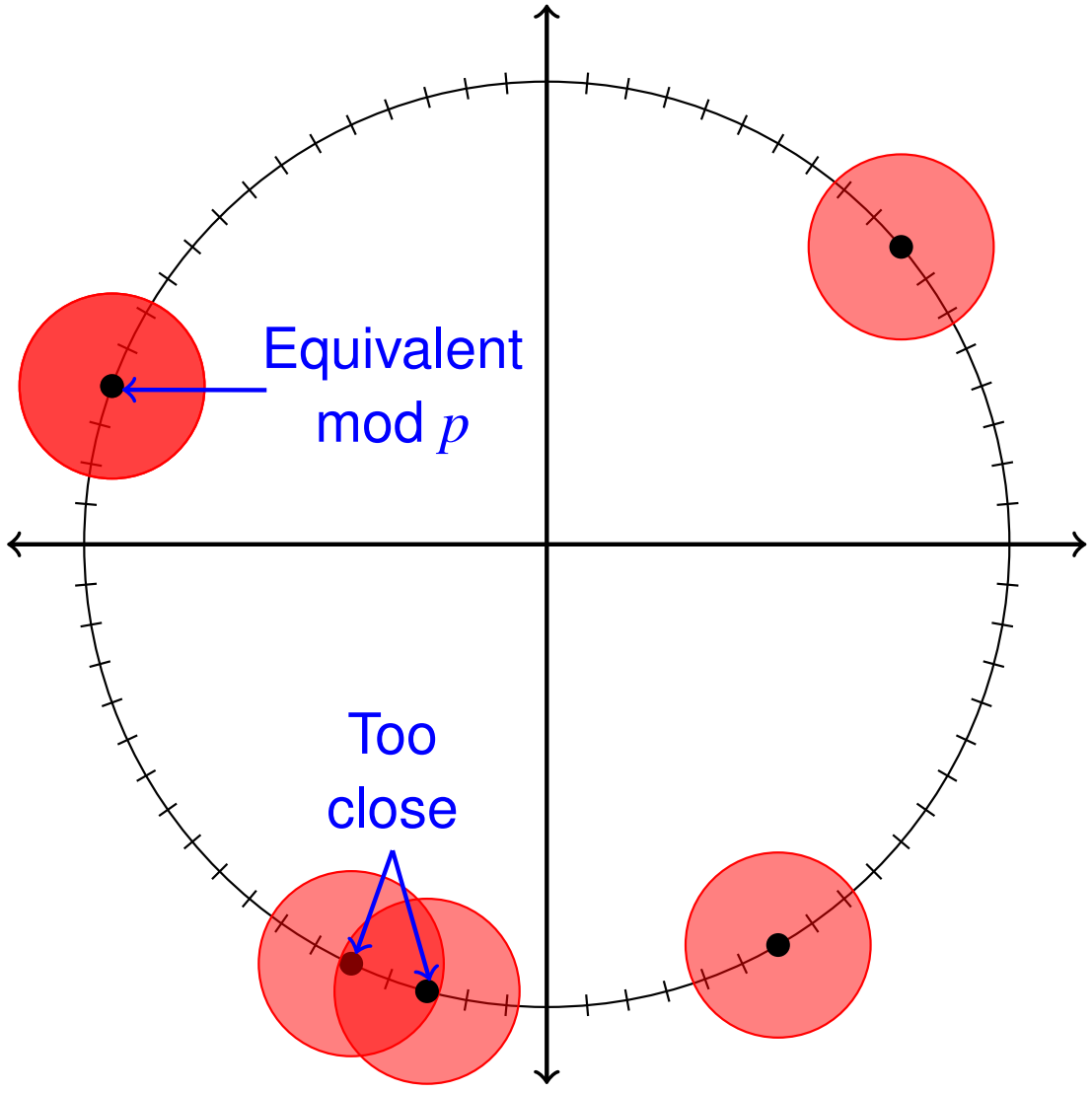
Embed de Prony's method inside our Garg & Schost-like method.

Instead of computing $f(1), f(\omega), \ldots f(\omega^{p-1})$ and using FFT,
we compute $f(1), f(\omega), \ldots, f(\omega^{2t-1})$ and use de Prony.

Diversifying the Exponents

- Need $\omega^{d_1}, \omega^{d_2}, \ldots, \omega^{d_t}$ to be sufficiently separated.

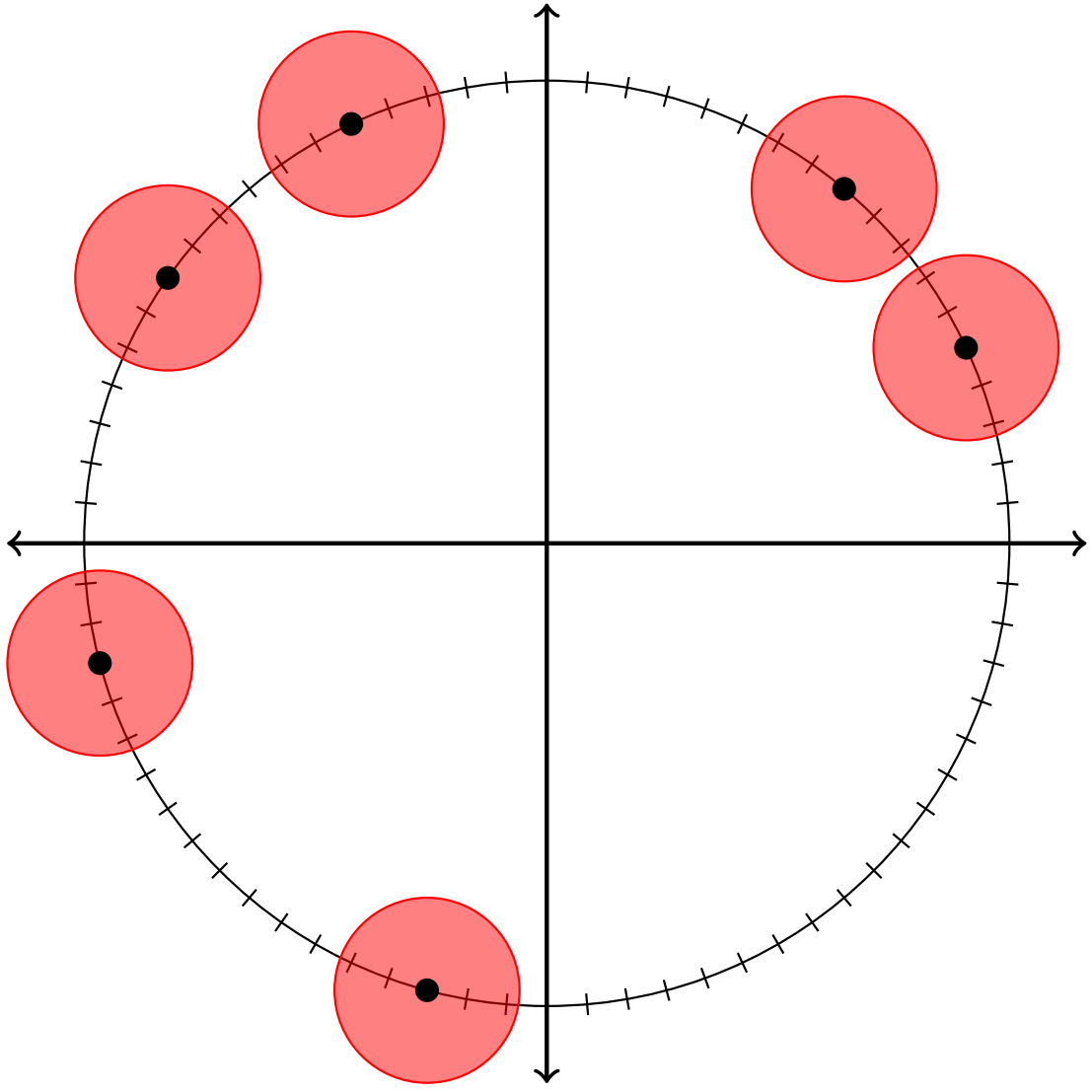
Bad choice of p **or** p-PRU ω:



Diversifying the Exponents

- Need $\omega^{d_1}, \omega^{d_2}, \ldots, \omega^{d_t}$ to be sufficiently separated.

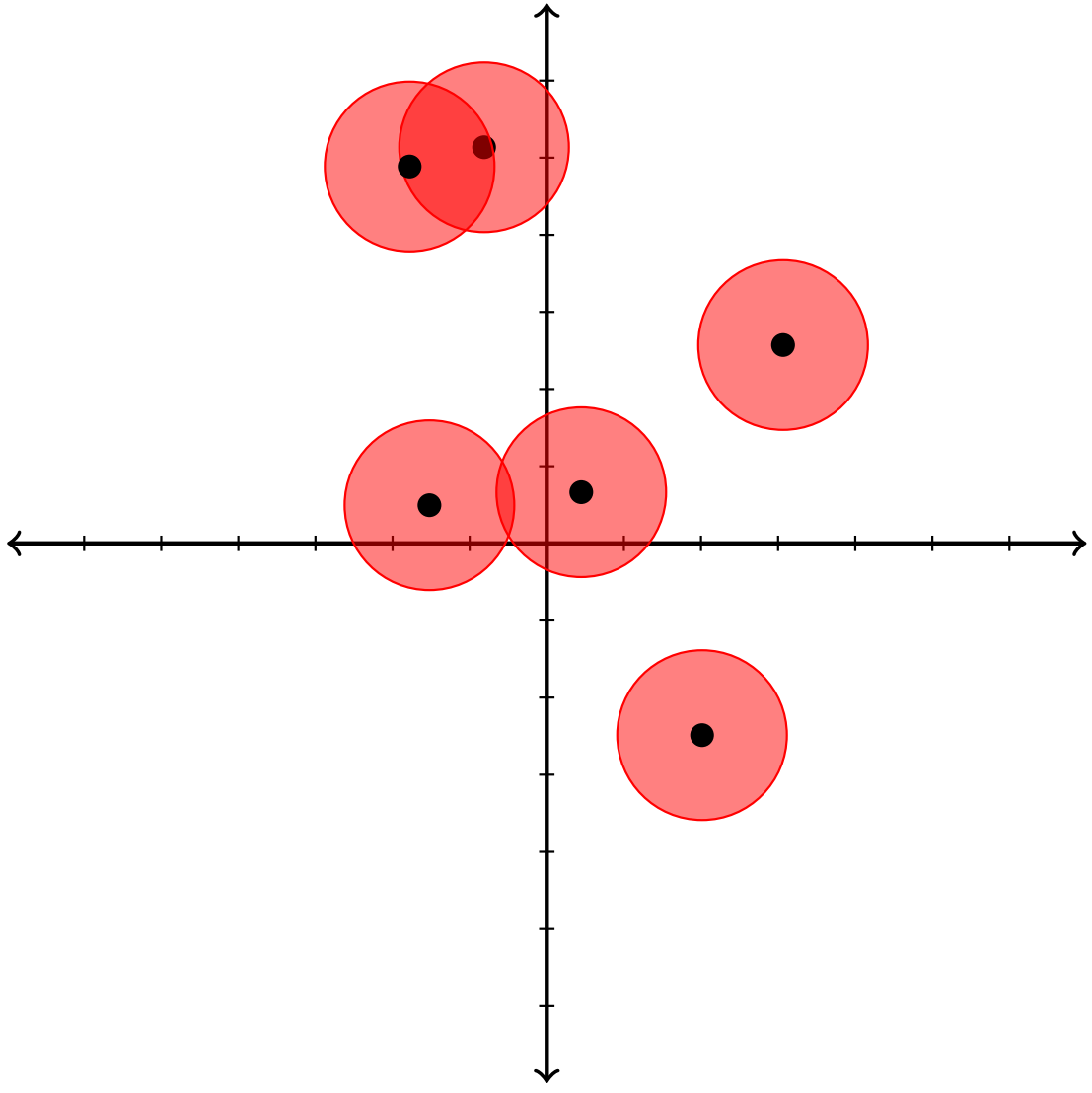
p sufficiently large, ω random p-PRU:



Diversifying the Coefficients

- Need $c_1, c_2, \ldots, c_t$ to be sufficiently distinct — impossible!

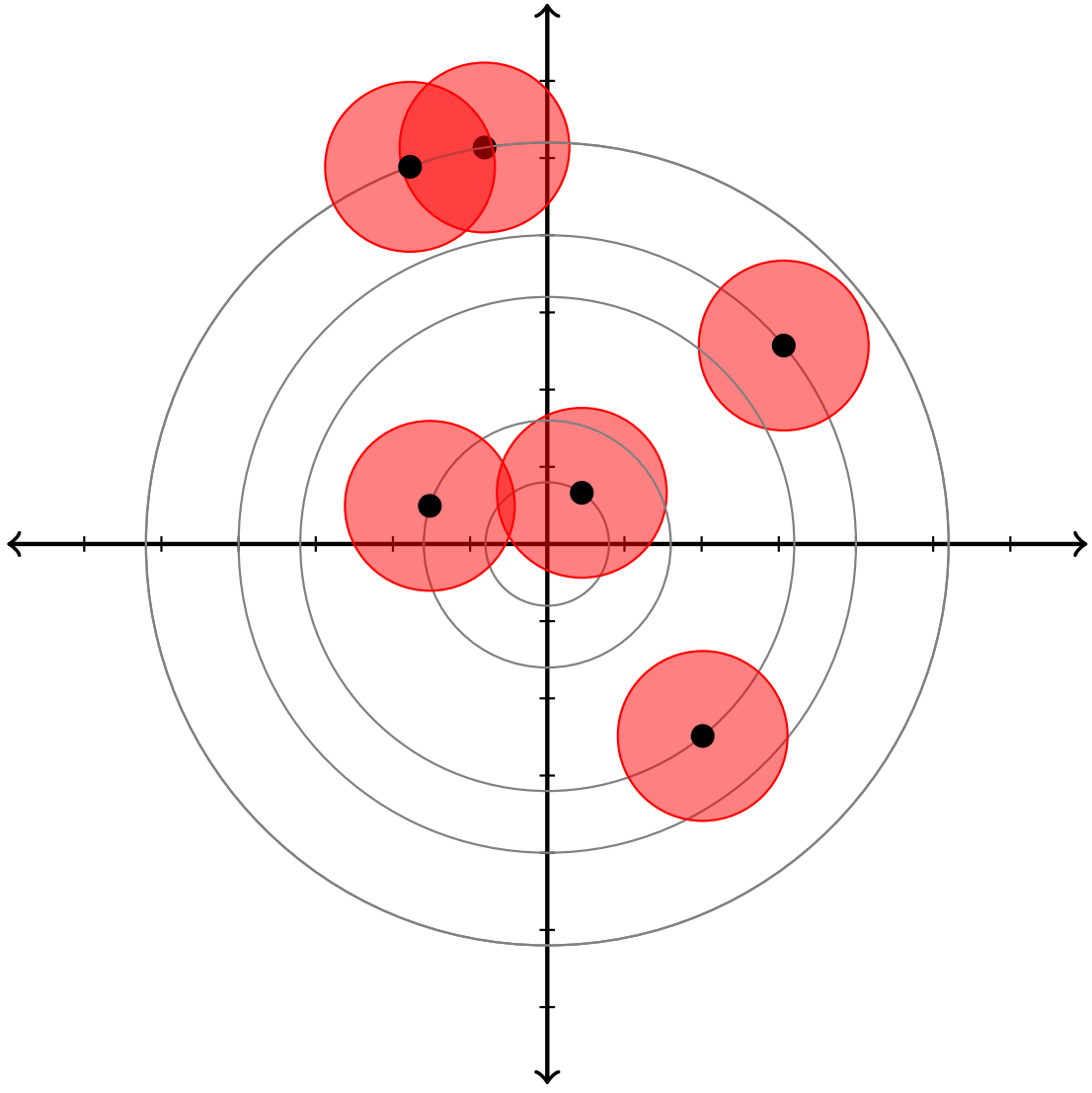
Not diverse:



Diversifying the Coefficients

- Need $c_1\zeta^{e_1}, c_2\zeta^{e_2}, \ldots, c_t\zeta^{e_t}$ to be sufficiently distinct

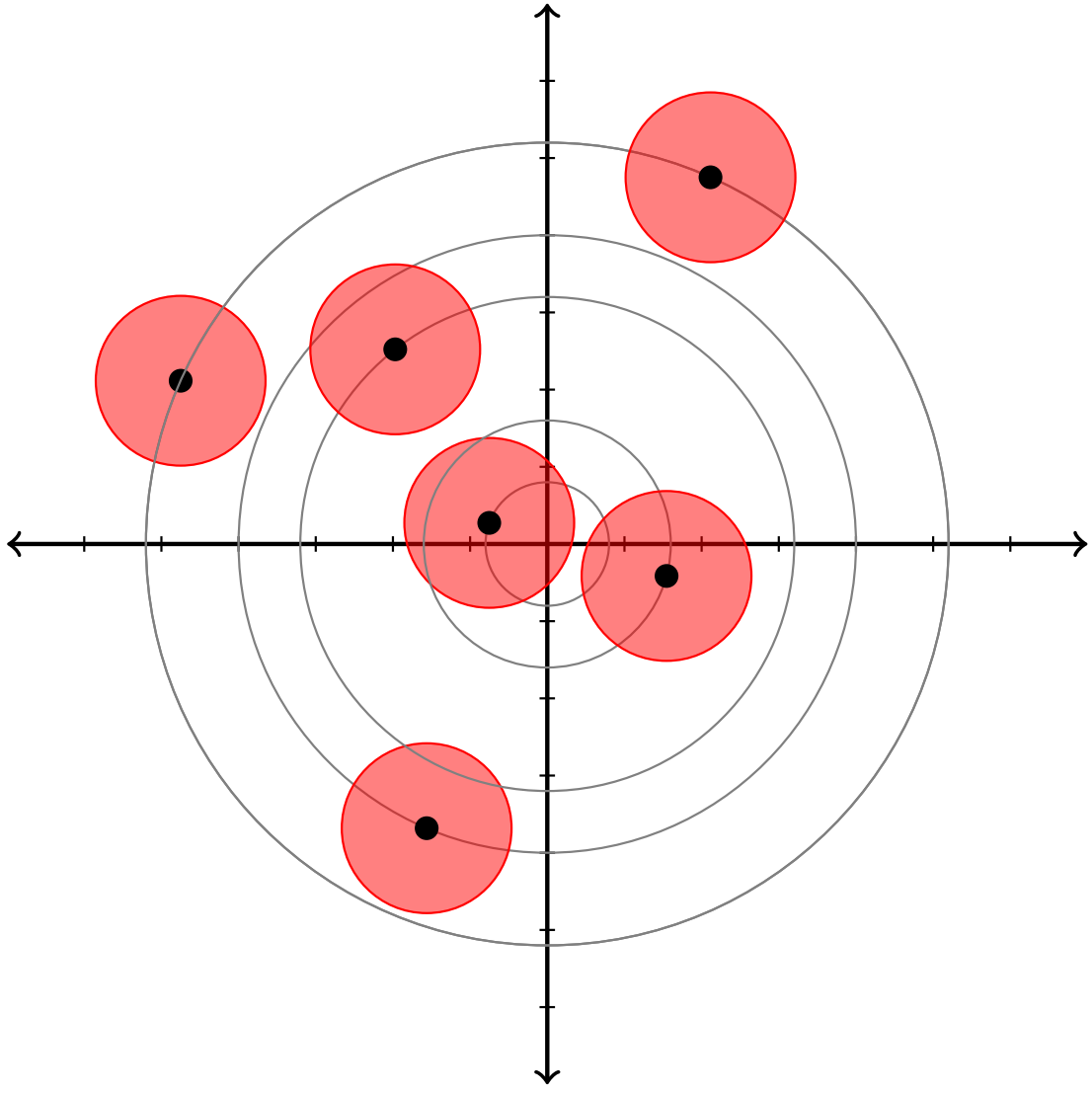
Bad choice of ζ:



Diversifying the Coefficients

- Need $c_1\zeta^{e_1}, c_2\zeta^{e_2}, \ldots, c_t\zeta^{e_t}$ to be sufficiently distinct

Good choice of ζ:



Overview of Combined Algorithm

1. Choose prime $s \in O(t^2)$ and random s-PRU ζ
2. Choose prime $p \in O(t^2 \log d)$ and random p-PRU ω
3. Evaluate $f(1), f(\zeta\omega), f(\zeta^2\omega^2), \ldots f(\zeta^{2t-1}\omega^{2t-1})$
4. Recover $f(\zeta x) \bmod (x^p - 1)$ using de Prony's method
5. Correlate coefficients with exponent residues modulo p
6. Repeat Steps 2–5 $O(\log d)$ times
7. Recover exponents from modular residues by Chinese remaindering

Overview of Combined Algorithm

1. Choose prime $s \in O(t^2)$ and random s-PRU ζ
2. Choose prime $p \in O(t^2 \log d)$ and random p-PRU ω
3. Evaluate $f(1), f(\zeta\omega), f(\zeta^2\omega^2), \ldots f(\zeta^{2t-1}\omega^{2t-1})$
4. Recover $f(\zeta x) \bmod (x^p - 1)$ using de Prony's method
5. Correlate coefficients with exponent residues modulo p
6. Repeat Steps 2–5 $O(\log d)$ times
7. Recover exponents from modular residues by Chinese remaindering

These are the randomization steps.

Results

Sparse interpolation algorithm featuring

- $O(t \log d)$ probes at (nearly) fixed precision
  — optimal in terms of total bit length
- $\tilde{O}(t^2 \log d)$ post-processing cost

Requires low precision, even at high degrees,
and with as few probes as possible.