Minimum enclosing circle
Problem: Given $n$ points in the plane, find min-radius circle containing them

May be either of two cases:

- Two points form its diameter
- Circle through an acute triangle
Why only those two cases?

Any other circle can be shrunk
Cannot be a linear program

In a $d$-dimensional linear program, optimal solution is determined by exactly $d$ constraints, but here solution sometimes comes from pairs of inputs and sometimes comes from triples.
Circle primitives

**Representation of circle**

center, \( \text{radius}^2 \)

**Does it contain a given point?**

compare point-to-center dist\(^2\) to radius\(^2\)

**Circle from two points**

center = average coords
radius\(^2\) = (dist\(^2\)/4)

**Circle from three points**

Construct lines through pairs
Rotate 90° at midpoints
They meet at circle center
radius\(^2\) = dist\(^2\) to any point
As a nonlinear program

Given points $x_i, y_i$:

Find center $X, Y$
and squared radius $R$

Obey nonlinear constraints
$$(x_i - X)^2 + (y_i - Y)^2 \leq R$$

Minimize linear objective $R$

Define $S = R - X^2 - Y^2$

Find $X, Y, S$ with linear constraints

$${x_i}^2 - 2x_i X + {y_i}^2 - 2y_i Y \leq S$$

Minimize nonlinear objective
$$S + X^2 + Y^2$$
LP-type problems
Key properties of circle problem

- We can describe it as a function mapping sets of points to their optimal solution \((X, Y, R)\).

- Monotonic: We can compare solutions, and if \(A \subset B\) are two sets of points then \(A\)'s solution is at least as good as \(B\)'s.

- Local: If adding \(p\) or \(q\) separately to a set doesn't change its solution, then neither does adding both at once.

- Low-dimensional: Every set has a basis of \(\leq 3\) points with same solution (diameter points or acute triangle).

- Primitives: Find circle defined by two or three points. "Violation test": is point inside circle?
Define a class of problems with the same properties:

- We can describe it as a function mapping sets of inputs to their optimal solution
- Monotonic: We can compare solutions, and if $A \subset B$ are two sets of inputs then $A$’s solution is at least as good as $B$’s
- Local: If adding $p$ or $q$ separately to a set doesn’t change its solution, then neither does adding both at once
- Low-dimensional: Every set has a basis of $O(1)$ points with same solution (‘‘dimension’’: maximum size of a basis)
- Primitives: Find solution for basis
  ‘‘Violation test’’: does adding $p$ to $B$ change its solution?

Goal: Find optimal solution using a small number of primitives
Linear programming is LP-type

Fix objective function, and consider subsets of constraints. Then:

▶ We can describe it as a function mapping sets of constraints to their optimal solution points and its objective value

▶ Monotonic: We can compare solutions by their objective value, and if $A \subset B$ are two sets of constraints then $A$’s solution is at least as good as $B$’s

▶ Local: If a solution point obeys constraints $p$ and $q$ when one of them is added to the set of constraints, it still obeys them when both of them are added.

▶ Low-dimensional: Every set has a basis of $d$ constraints with same solution

▶ Primitives: Solve basis using Gaussian elimination

Violation test: compute linear inequality on solution point
More example problems
Closest distance between disjoint convex hulls

Map subsets to hull distance
\(= \max \) separation of parallel lines between hulls

Monotonic: More points \(\Rightarrow\) closer hulls

Local: if two points stay outside parallel lines when added separately, they stay outside when added together

Low-dim: In \(\mathbb{R}^d\), solution determined by \(\leq d + 1\) points

Like red-blue separation but Euclidean not vertical distance
A problem that is not LP-type

Enclose $n$ points between parallel lines as close together as possible.

Like $L_\infty$ regression, but Euclidean distance not vertical distance.

It is monotonic and local.

For input $=$ regular $(2n + 1)$-sided polygon, optimal solutions use lines through one side and opposite vertex.

If any point is removed, solution gets better.

So all $2n + 1$ points are needed to determine the solution.

$\Rightarrow$ not low-dimensional.
Algorithms
Seidel’s algorithm

If we already know that certain inputs must be in the optimal basis, we can keep them in a separate set $B$, reducing the LP-type-dimension of the remaining problem by $|B|$

Call the following with Seidel(input set, empty set):

```python
def Seidel(X,B):
    Compute $S = \text{solution}(B)$
    Done = empty set
    For each element $x$ of $X$ in random order:
        If $x$ fails violation test for $S$:
            $S = \text{Seidel}(\text{Done}, B \cup \{x\})$
            Add $x$ to Done
    Return $S$
```
Choose and solve a random subset $R$ of the items

Let $V(R) =$ items that fail violation test for solution

- Each $x$ in $V$ must be part of basis for $V \cup \{x\}$
- Probability that this is true for $x$ is $\leq d/(|R| + 1)$
- So expected size of $V(R)$ is $O(n/|R|)$
- If $V(R)$ is non-empty, it includes at least one member of basis of whole problem
Recursive sampling

\( V = \text{empty set} \)
repeat \( d \) times:
  Choose sample \( R \) of size \( \sqrt{n} \)
  Compute solution for \( R \cup V \)
  Add its violators to \( V \)

Each time through the loop, adds \( O(\sqrt{n}) \) elements to \( V \) including at least one more basis element

Solves whole problem in \( O(dn) \) violation tests and \( O(d) \) recursive calls on problems of size \( O(d\sqrt{n}) \)
Iterated reweighting

Give all elements weight 1
Repeat:

Select a random subset of $2d$ elements, with probability proportional to their weights
Compute solution for subset and its set $V$ of violated elements
Double the weights of the members of $V$

Total expected weight of all items increases by $(1 + 1/2d)$ factor
Weight of optimal basis increases by larger $(1 + 1/d)$ factor
Can only happen $O(d \log n)$ times until subset has bigger weight than whole set (impossible) or we find optimal solution

Solves whole problem in $O(dn \log n)$ violation tests and $O(d \log n)$ recursive calls on problems of size $O(d)$
Putting it together

Outer level of algorithm: use random sampling
- $O(dn)$ violation tests
- $O(d)$ second-level calls on subproblems of size $O(d\sqrt{n})$

Second-level calls: use iterated reweighting
- Each subproblem has size $O(d\sqrt{n})$
- It does $O(d^2\sqrt{n}\log n)$ violation tests
- $O(d\log n)$ third-level calls on subproblems of size $O(d)$

Third-level calls: use Seidel’s algorithm
- Each subproblem has size $O(d)$
- $d^{O(d)}$ solution primitives

Total: $O(dn)$ violation tests, $d^{O(d)}\log n$ solution primitives
Can reduce $d^{O(d)}$ to $d^{O(\sqrt{d})}$ by replacing Seidel