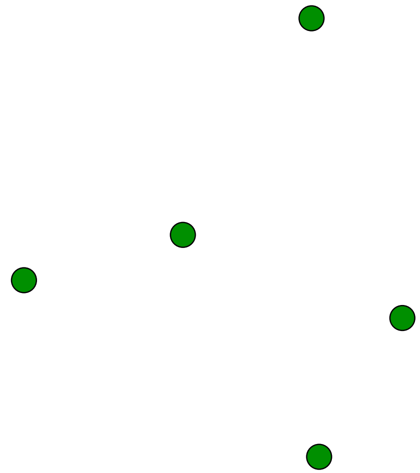
Fast Smallest-Enclosing-Ball Computation in High Dimensions

Martin Kutz
FU Berlin

joint work with Bernd Gärtner and Kaspar Fischer, ETH Zürich

The Smallest Enclosing Ball

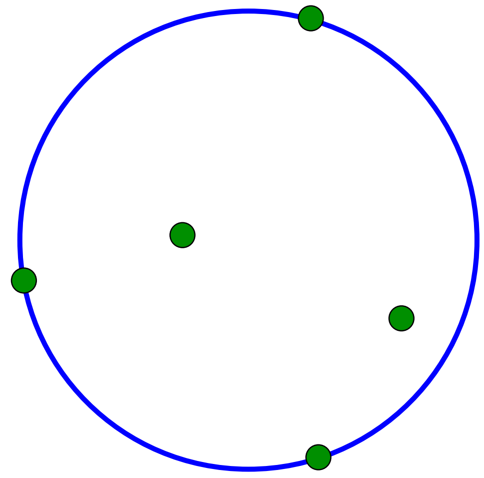
given: finite point set $\color{green}{S}$ in $\mathbb{R}^d$



The Smallest Enclosing Ball

given: finite point set S in $\mathbb{R}^d$

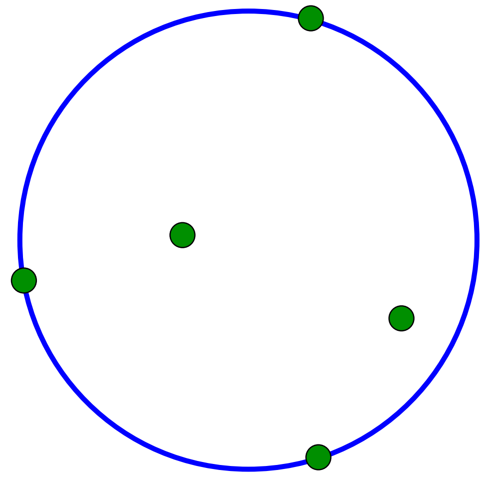
wanted: smallest ball B
$= B(c, r) := \{x : \|x - c\| \leq r\}$
containing S



The Smallest Enclosing Ball

given: finite point set S in $\mathbb{R}^d$

wanted: smallest ball B
$= B(c, r) := \{x : \|x - c\| \leq r\}$
containing S



Call this unique minimal B the *smallest enclosing ball* of S, denoted $\mathrm{seb}(S)$.

Applications

- visibility culling and bounding sphere hierarchies in 3D computer graphics
- clustering (e.g. for support-vector machines) — many dimensions
- nearest neighbor search

Previous Work

- Welzl proposed randomized combinatorial algorithm, implemented by Gärtner, fast for $d \leq 30$, impractical above.

- Quadratic-programming approach by Gärtner & Schönherr, uses exact arithmetic, up to $d = 300$.

Previous Work

- Welzl proposed randomized combinatorial algorithm, implemented by Gärtner, fast for $d \leq 30$, impractical above.

- Quadratic-programming approach by Gärtner & Schönherr, uses exact arithmetic, up to $d = 300$.

- General-purpose QP-solver CPLEX, solves $d \leq 3{,}000$.

Previous Work

- Welzl proposed randomized combinatorial algorithm, implemented by Gärtner, fast for $d \leq 30$, impractical above.
- Quadratic-programming approach by Gärtner & Schönherr, uses exact arithmetic, up to $d = 300$.
- General-purpose QP-solver CPLEX, solves $d \leq 3{,}000$.
- Zhou, Toh, and Sun use interior-point method to find approximate solution, up to $d = 10{,}000$.
- Kumar, Mitchell, Yildrum compute approximation with core sets, results given up to $d = 1{,}400$.

Our Algorithm

- simple *combinatorial* algorithm (not approximation)

Our Algorithm

- simple *combinatorial* algorithm (not approximation)
- similar to LP simplex-method
- equipped with pivot scheme to avoid cycling

Our Algorithm

- simple *combinatorial* algorithm (not approximation)
- similar to LP simplex-method
- equipped with pivot scheme to avoid cycling
- C++ floating-point implementation: solves several thousand points in a few thousand dimensions

Our Algorithm

- simple *combinatorial* algorithm (not approximation)
- similar to LP simplex-method
- equipped with pivot scheme to avoid cycling
- C++ floating-point implementation: solves several thousand points in a few thousand dimensions
- idea not completely new; Hopp & Reeve presented similar algorithm but without proofs, some details unclear, 3D only

The Basic Idea: Deflating a Ball

Iteratively shrink an enclosing Ball $B = B(c, T)$ represented by

- a current center c,
- an affinely independent subset $T \subseteq S$ of points at a common distance from c — the *support set*

The Basic Idea: Deflating a Ball

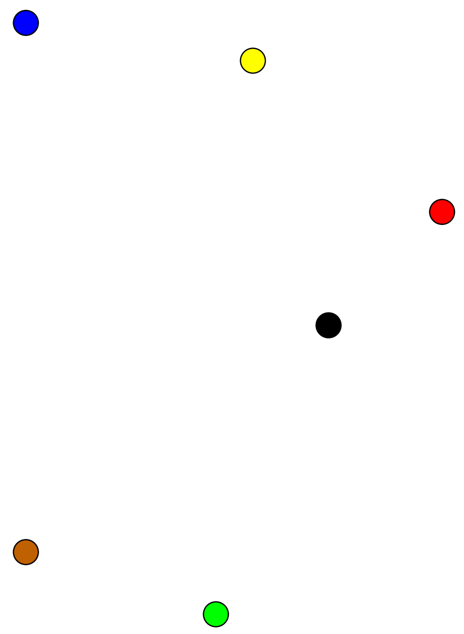
Iteratively shrink an enclosing Ball $B = B(c, T)$ represented by

- a current center c,
- an affinely independent subset $T \subseteq S$ of points at a common distance from c — the *support set*

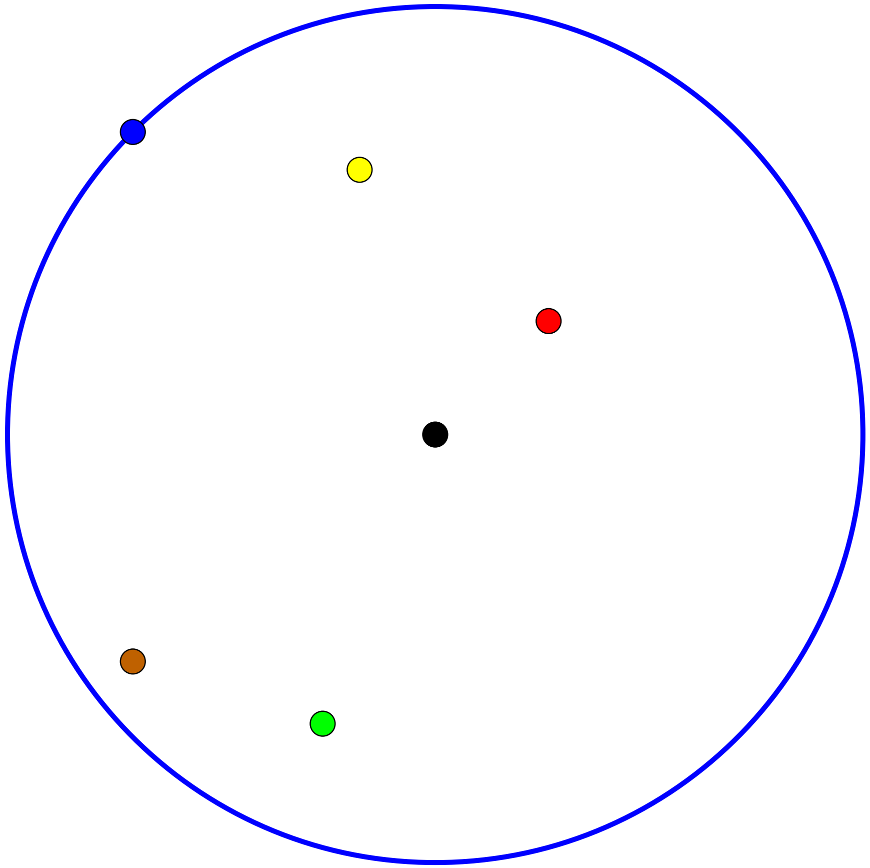
Invariants:

$$\begin{aligned} T &\subset \partial B(c, T) \\ S &\subset B(c, T) \end{aligned}$$

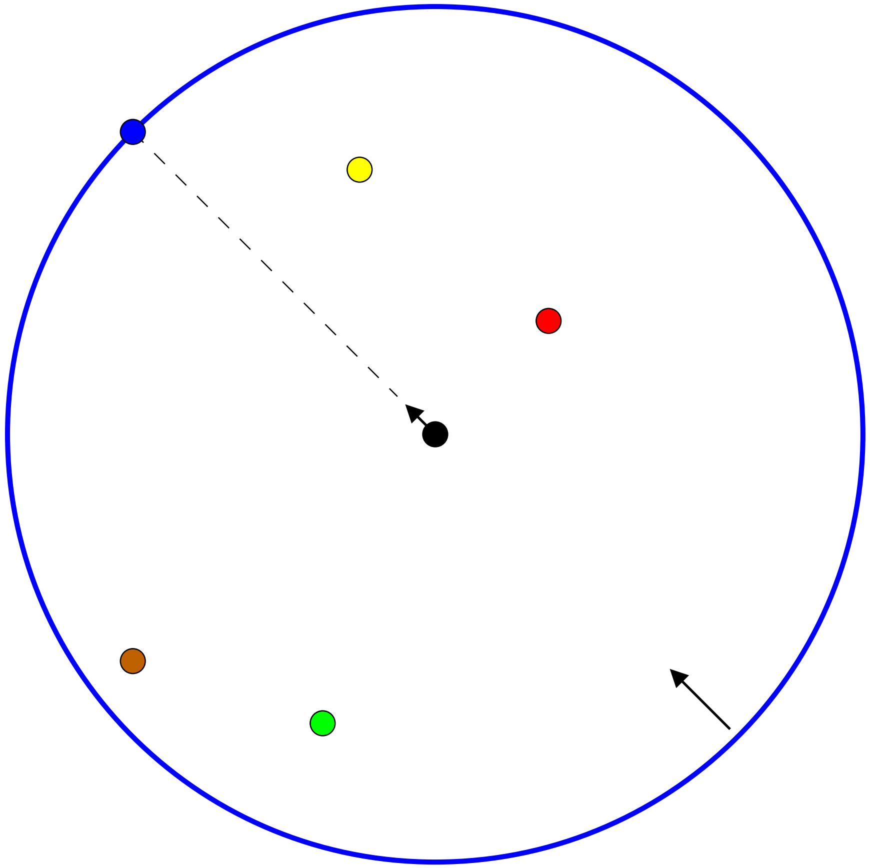
2D "Example"



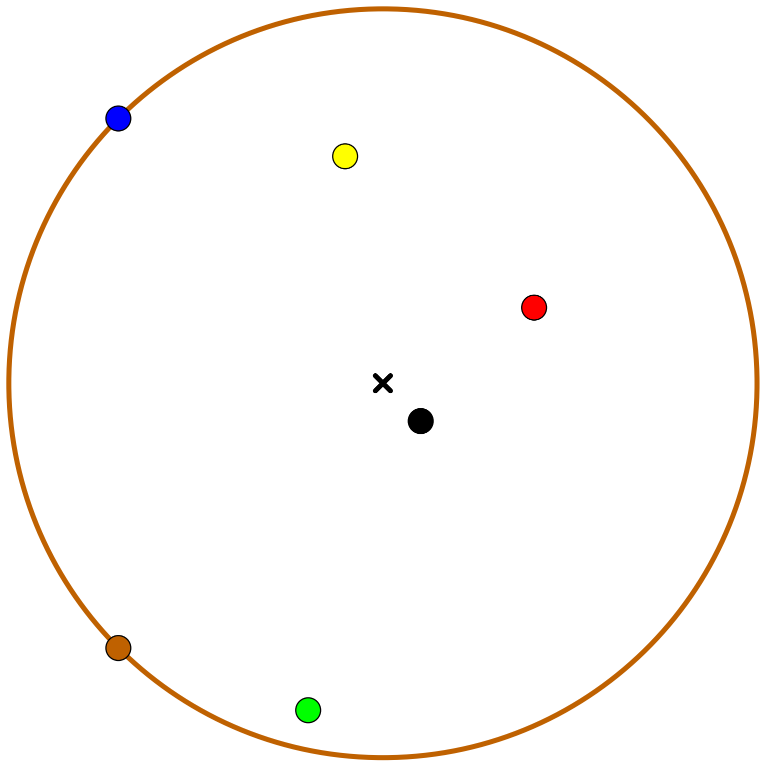
2D “Example”



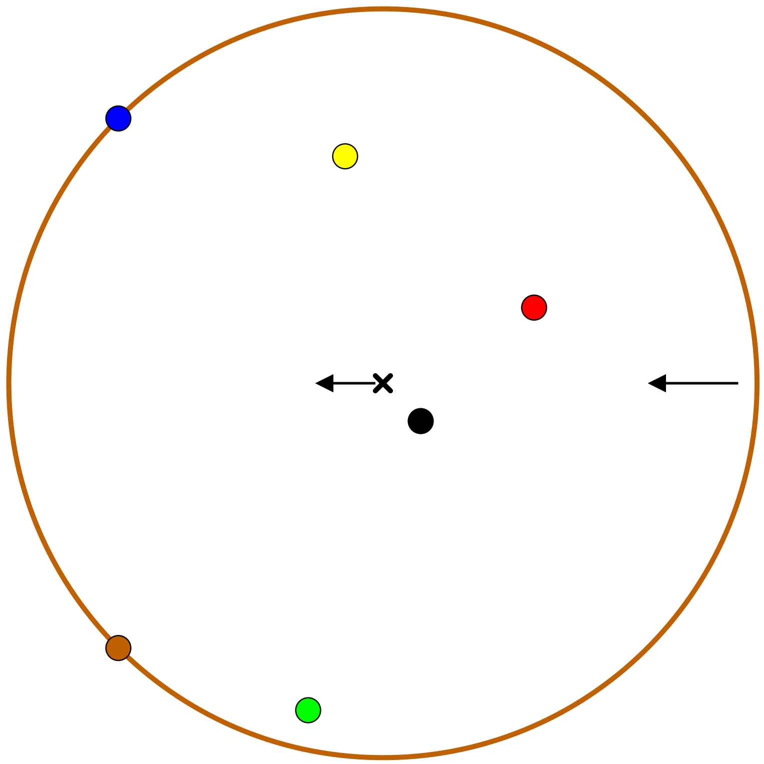
2D “Example”



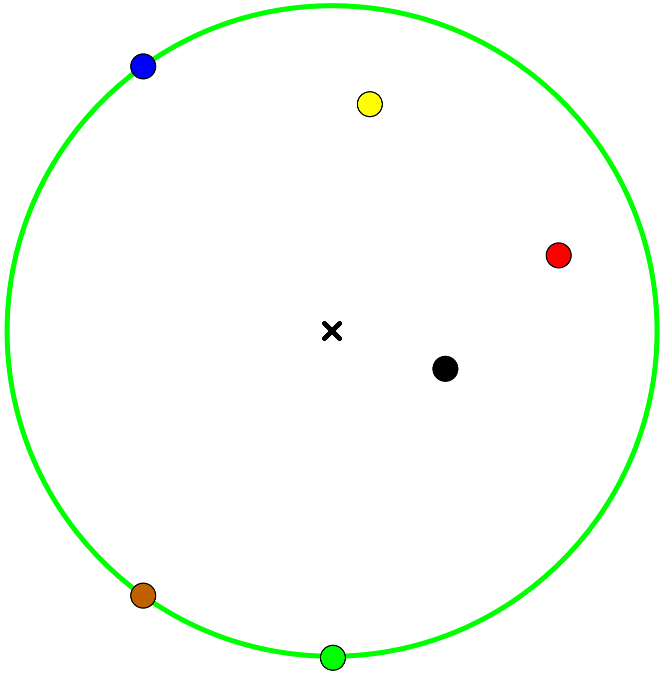
2D "Example"



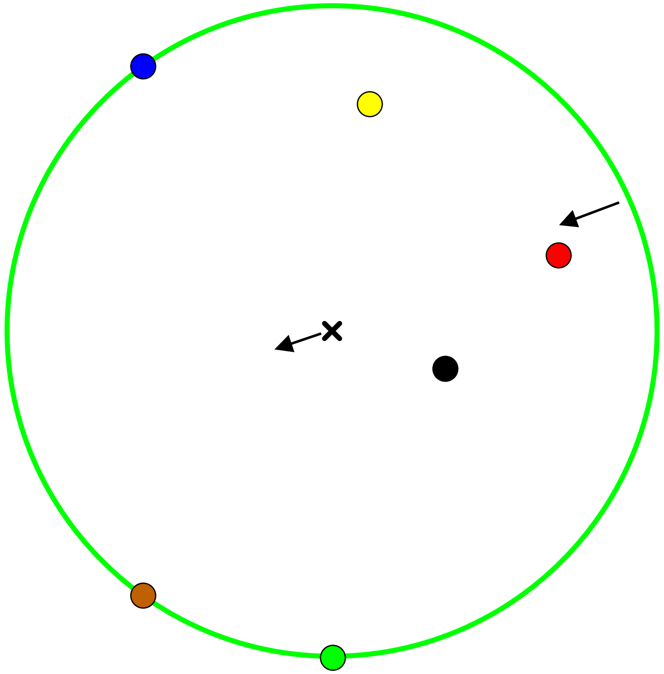
2D "Example"



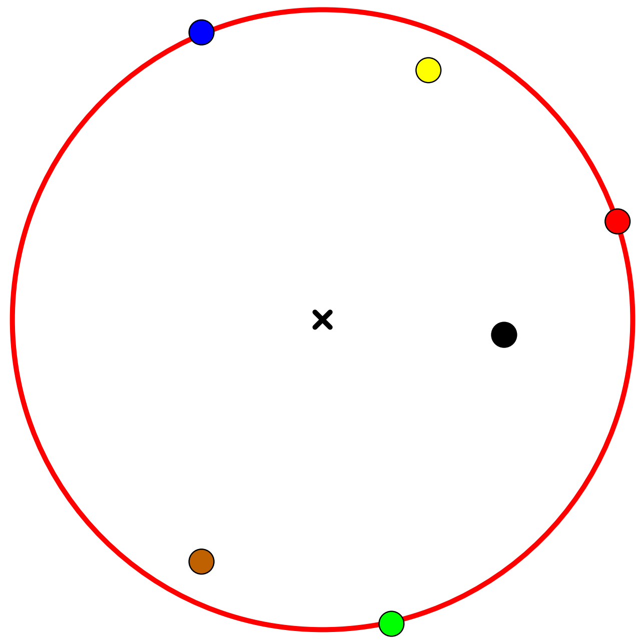
2D "Example"



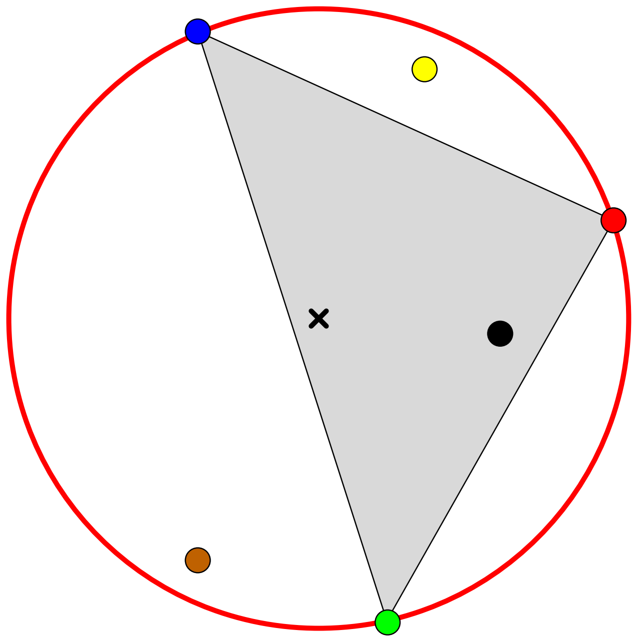
2D "Example"



2D "Example"



Termination Criterion



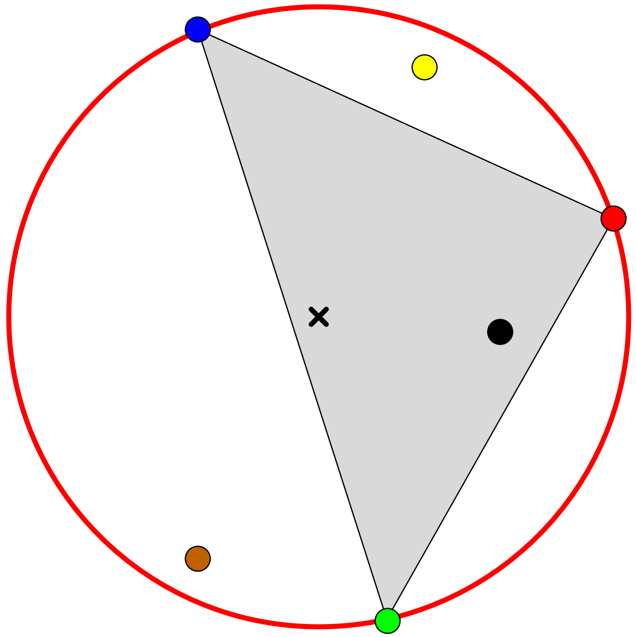
Termination Criterion

Lemma (Seidel).

Let T be set of points on boundary of some ball B with center c.

Then

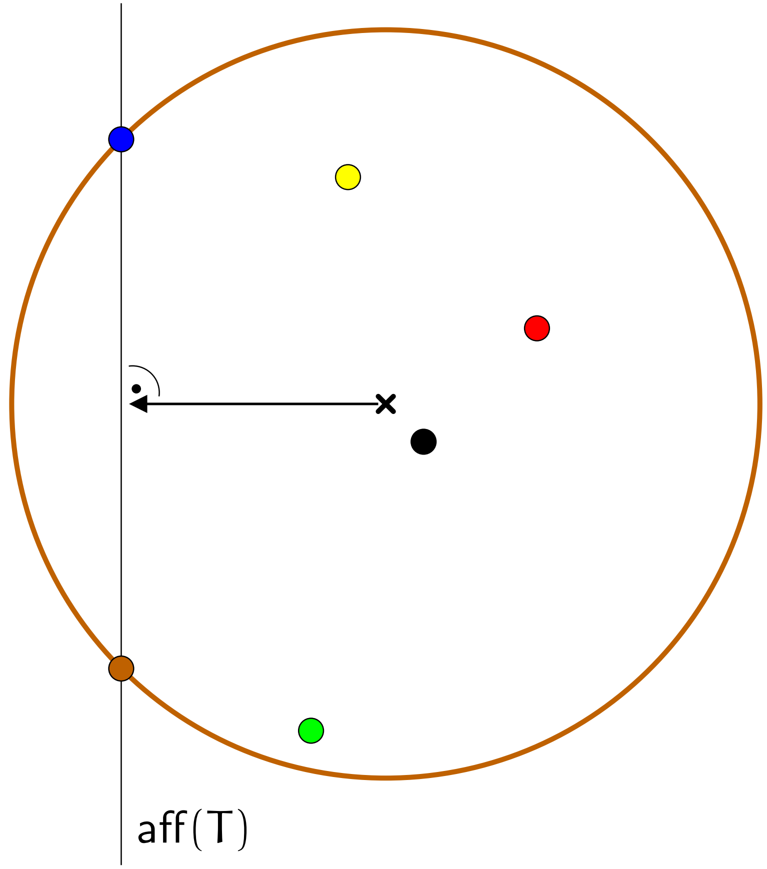
$$B = \mathrm{seb}(T) \quad \Longleftrightarrow \quad c \in \mathrm{conv}(T).$$



How to Shrink

Moving Step [Precondition $c \notin \mathrm{aff}(T)$]

Move c orthogonally towards $\mathrm{aff}(T)$,
i.e., heading for closest point in $\mathrm{aff}(T)$.



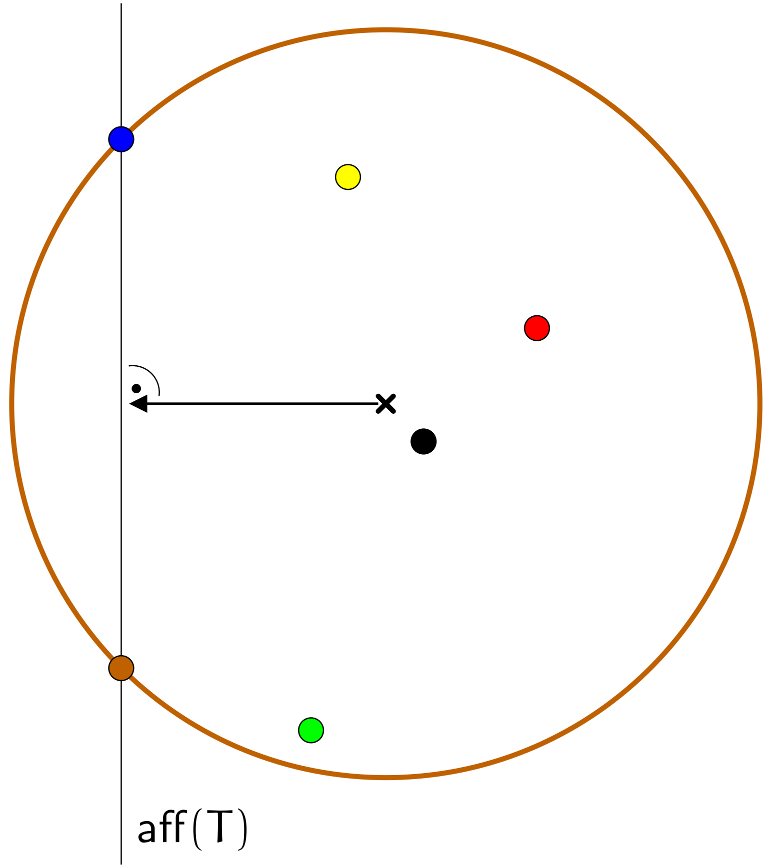
How to Shrink

Moving Step [Precondition $c \notin \text{aff}(T)$]

Move c orthogonally towards $\text{aff}(T)$,
i.e., heading for closest point in $\text{aff}(T)$.

For any fixed point c' on this path,
T stays on sphere around c'.

Especially, our target point is the center of the unique sphere through T in $\text{aff}(T)$, called circumcenter of T.



How to Shrink

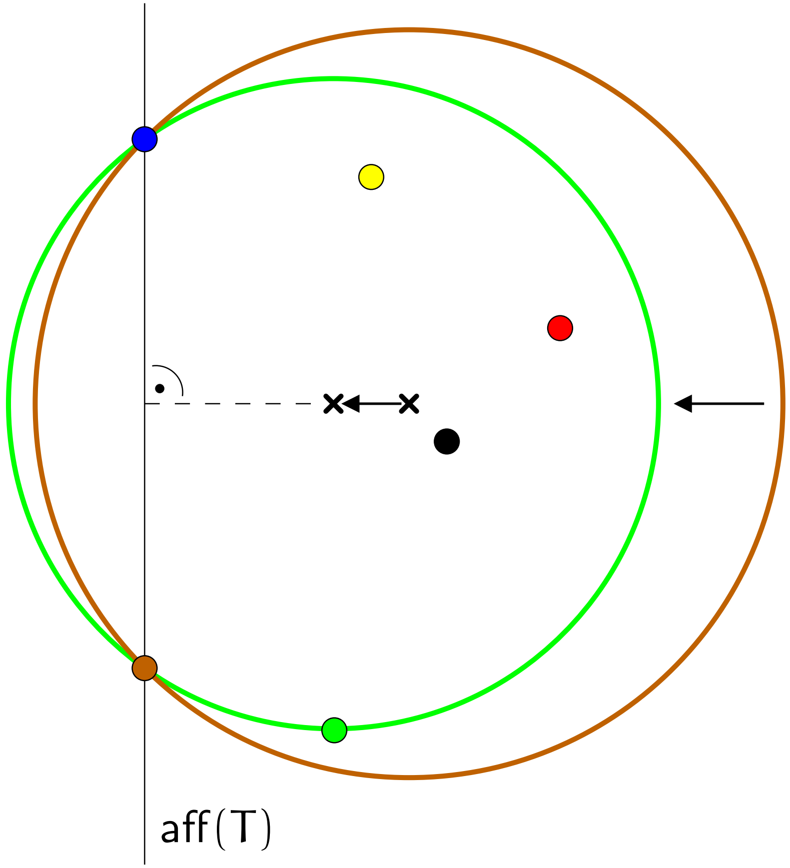
Moving Step [Precondition $c \notin \mathrm{aff}(T)$]

Move c orthogonally towards $\mathrm{aff}(T)$,
i.e., heading for closest point in $\mathrm{aff}(T)$.

For any fixed point c' on this path,
T stays on sphere around c'.

Especially, our target point is the
center of the unique sphere through T
in $\mathrm{aff}(T)$, called circumcenter of T.

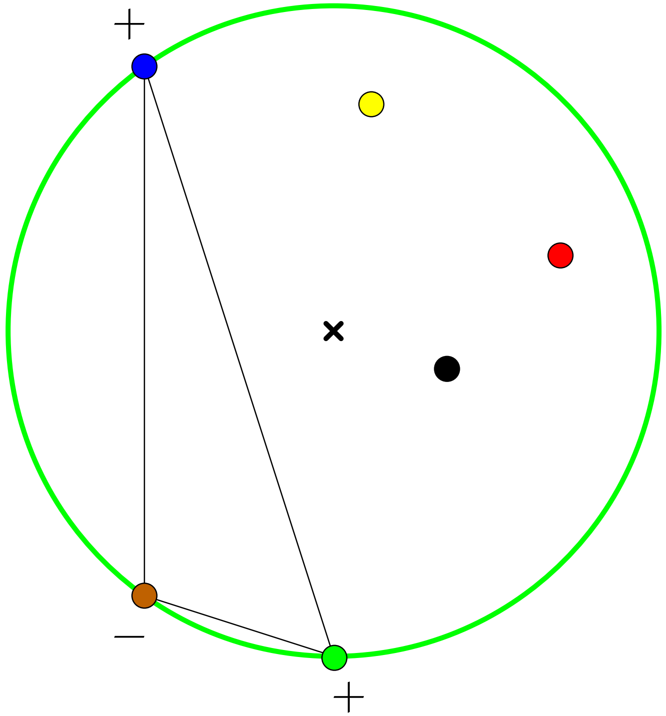
Stop movement when shrinking boundary
hits new point of S, insert it into T;
otherwise just stop with c in $\mathrm{aff}(T)$.



Dropping Step

Necessary if $c \in \mathrm{aff}(T) \setminus \mathrm{conv}(T)$.

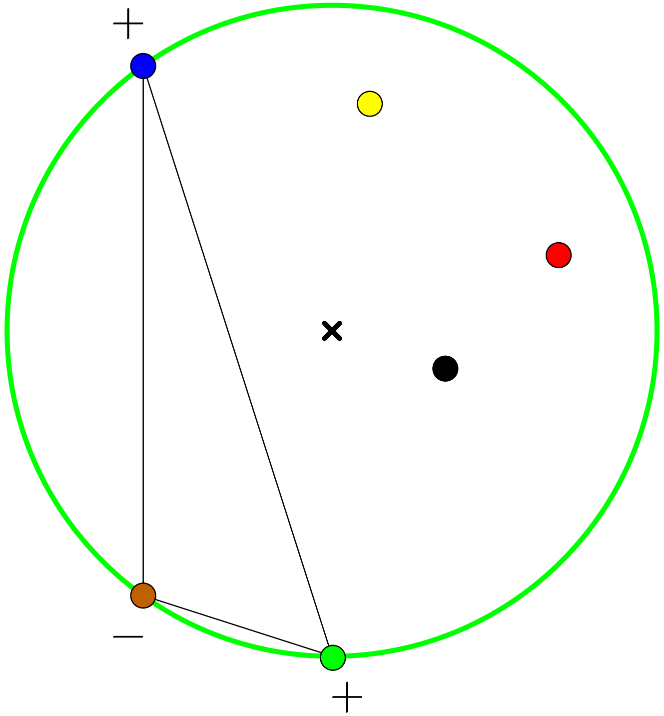
Must remove a point from T.



Dropping Step
Necessary if $c \in \mathrm{aff}(T) \setminus \mathrm{conv}(T)$.

Must remove a point from T.
Pick one with negative coefficient in affine representation

$$c = \sum_{p \in T} \lambda_p p, \quad \sum_{p \in T} \lambda_p = 1.$$

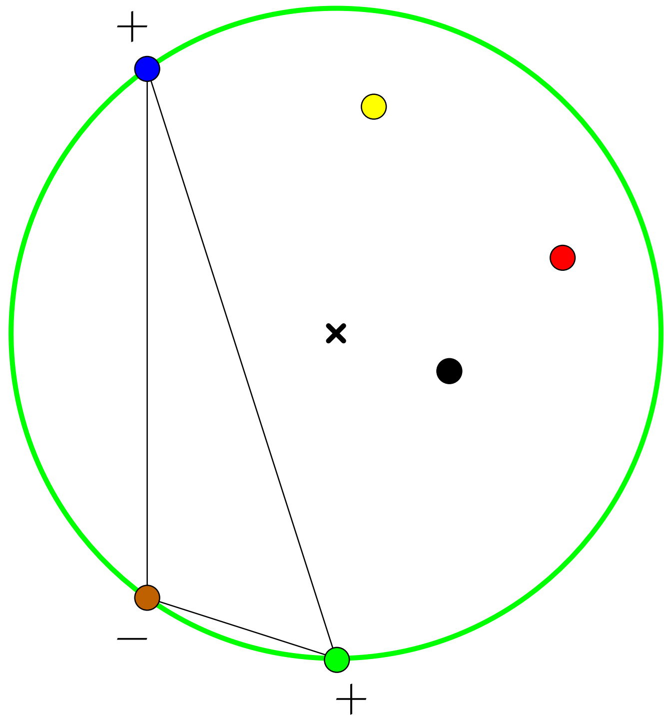


Dropping Step
Necessary if $c \in \mathrm{aff}(T) \setminus \mathrm{conv}(T)$.

Must remove a point from T.
Pick one with negative coefficient in affine representation

$$c = \sum_{p \in T} \lambda_p p, \quad \sum_{p \in T} \lambda_p = 1.$$

Afterwards, c lies outside the new $\mathrm{aff}(T)$, so it we can move again.
The next move will *not* recollect the dropped point.



The Whole Algorithm

$\mathsf{c} :=$ any point of S;
$\mathsf{T} := \{p\}$, with some $p \in S$ at maximal distance from c;

while $\mathsf{c} \notin \mathsf{conv}(\mathsf{T})$ **do**
  [Invariant: $B(\mathsf{c}, \mathsf{T}) \supset \mathsf{S}$, $\partial B(\mathsf{c}, \mathsf{T}) \supset \mathsf{T}$, and T affinely independent]

  if $\mathsf{c} \in \mathsf{aff}(\mathsf{T})$ **then** drop T-point with negative coefficient in aff. rep. of c;
  [Invariant: $\mathsf{c} \notin \mathsf{aff}(\mathsf{T})$]

The Whole Algorithm

$c :=$ any point of S;
$T := \{p\}$, with some $p \in S$ at maximal distance from c;

while $c \notin \mathrm{conv}(T)$ **do**

[Invariant: $B(c,T) \supset S$, $\partial B(c,T) \supset T$, and T affinely independent]

if $c \in \mathrm{aff}(T)$ **then** drop T-point with negative coefficient in aff. rep. of c;

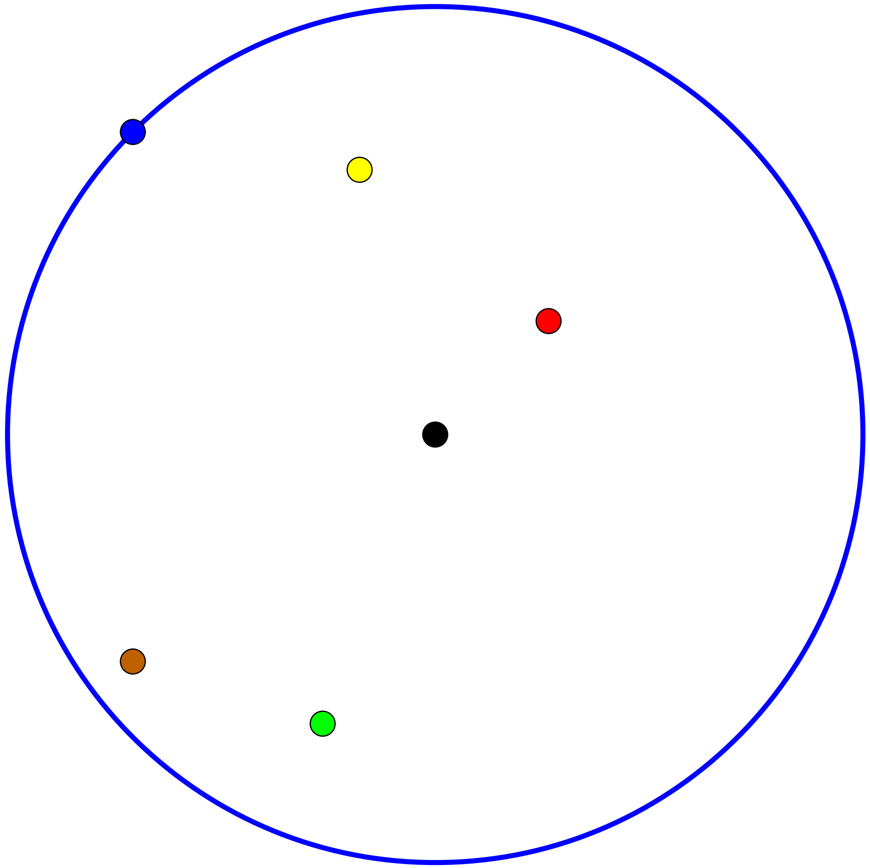
[Invariant: $c \notin \mathrm{aff}(T)$]

move c towards $\mathrm{aff}(T)$,
stop when boundary hits new point $q \in S$ or c reaches $\mathrm{aff}(T)$;

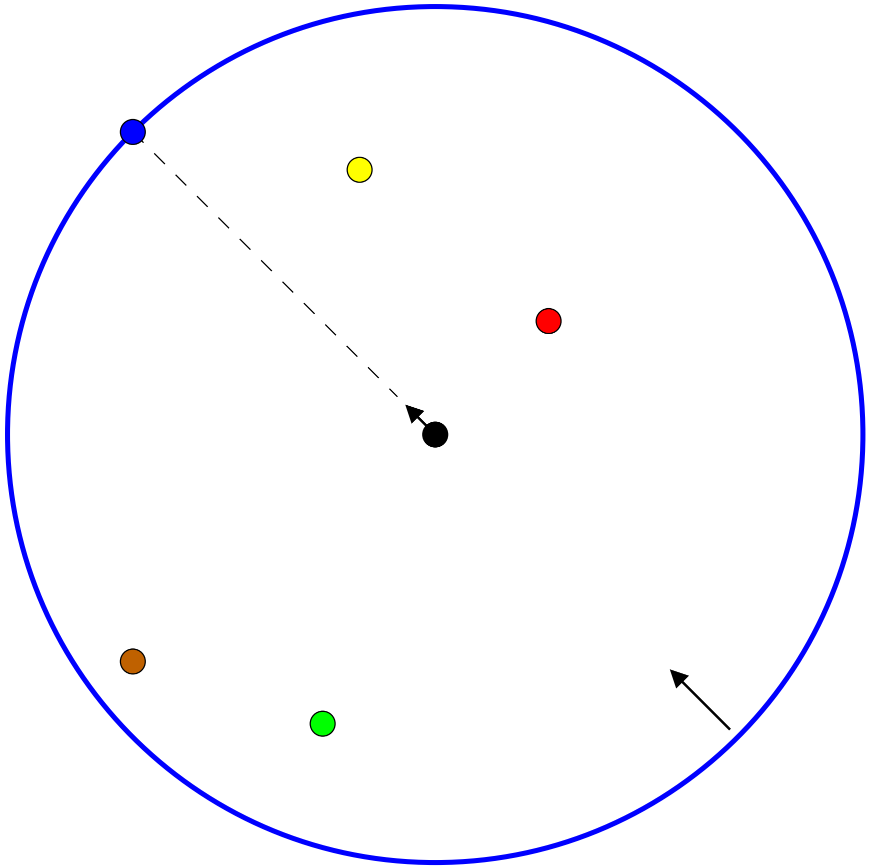
if point stopped us **then** $T := T \cup \{q\}$;

end while;

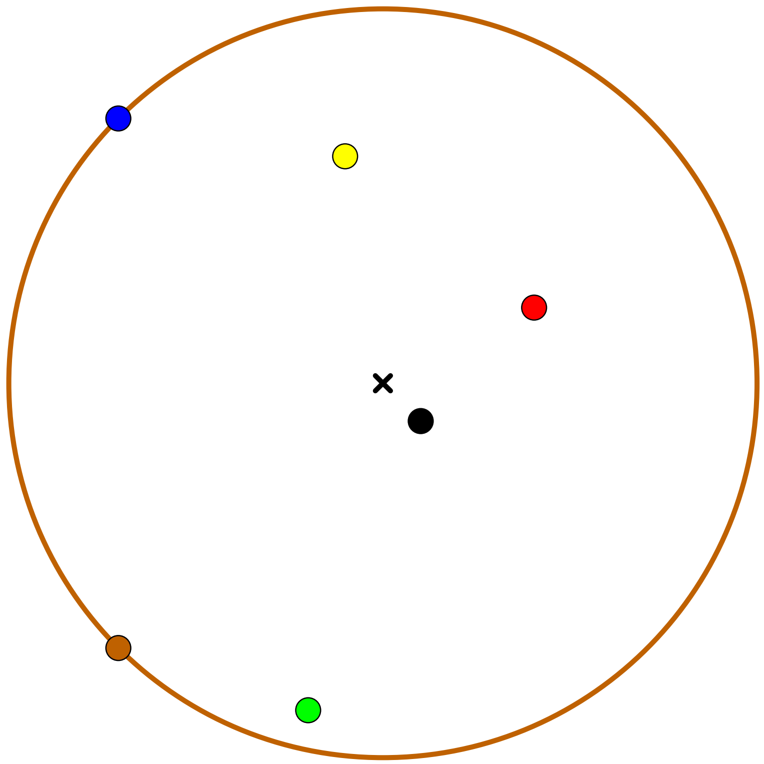
Our Example Again



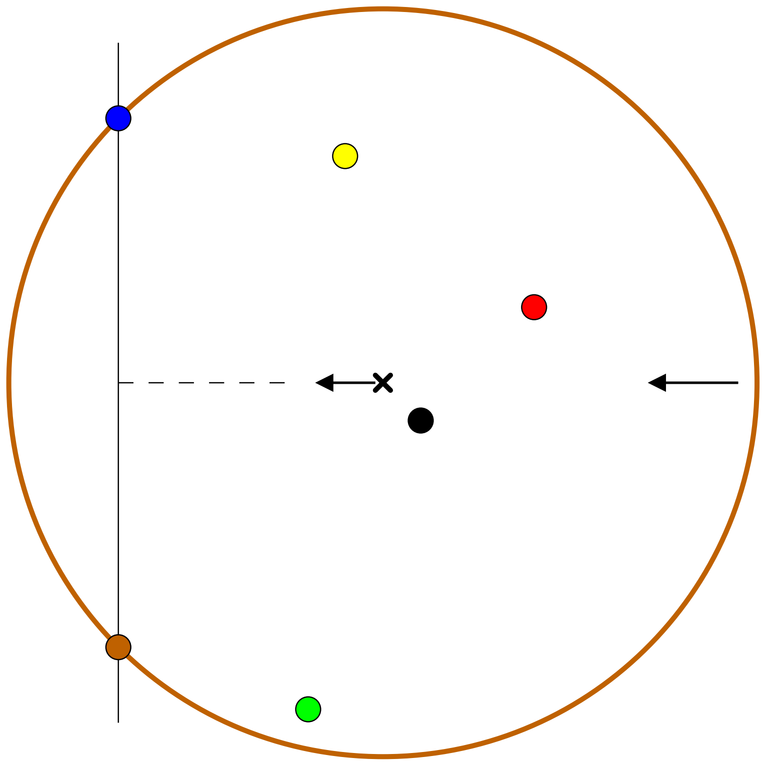
Our Example Again



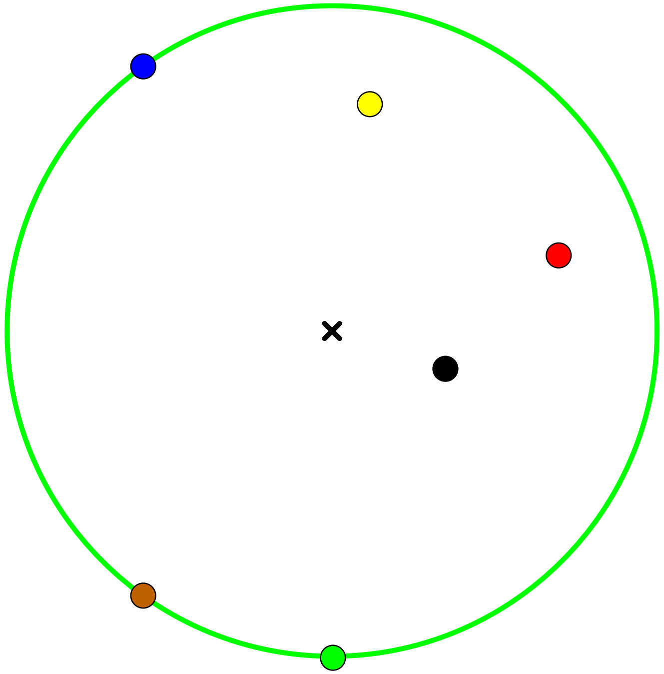
Our Example Again



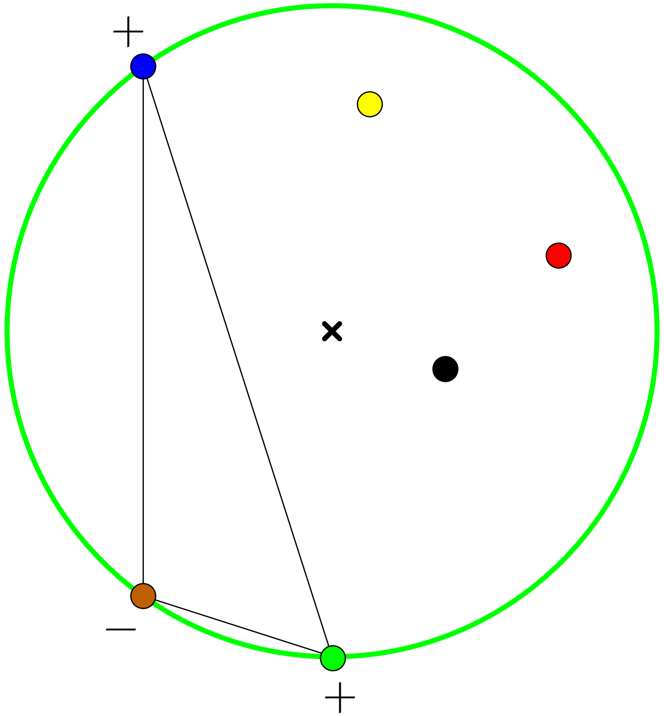
Our Example Again



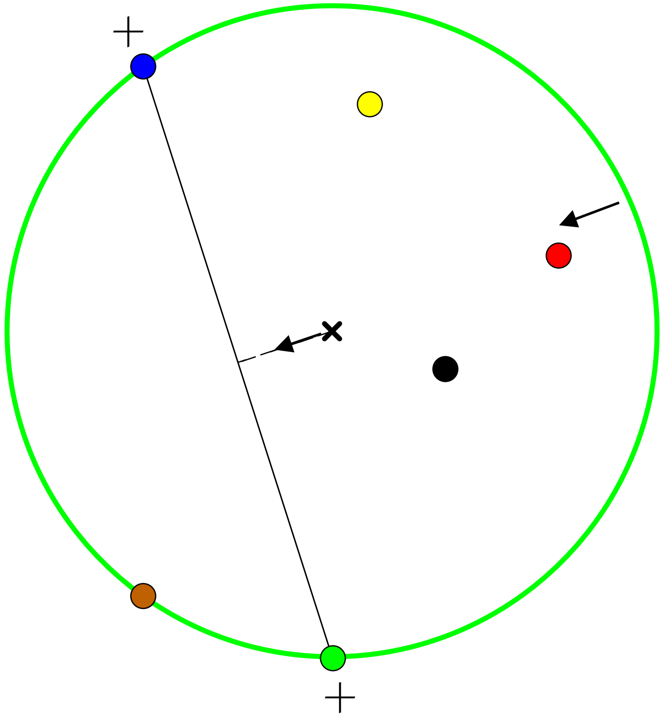
Our Example Again



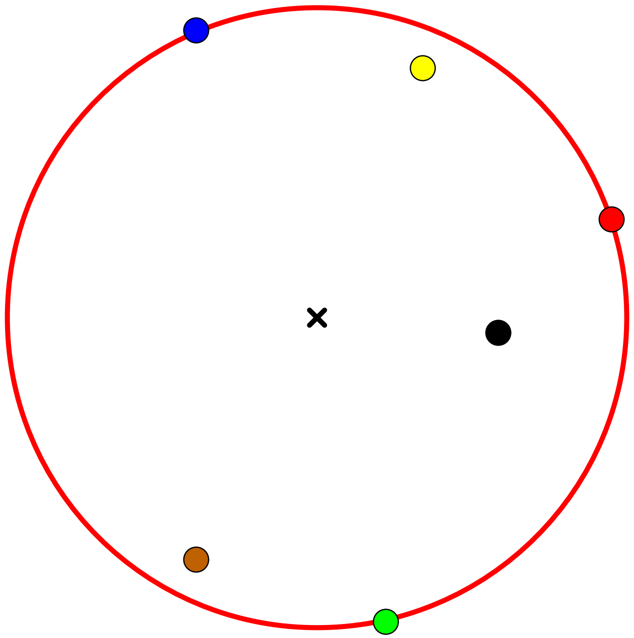
Our Example Again



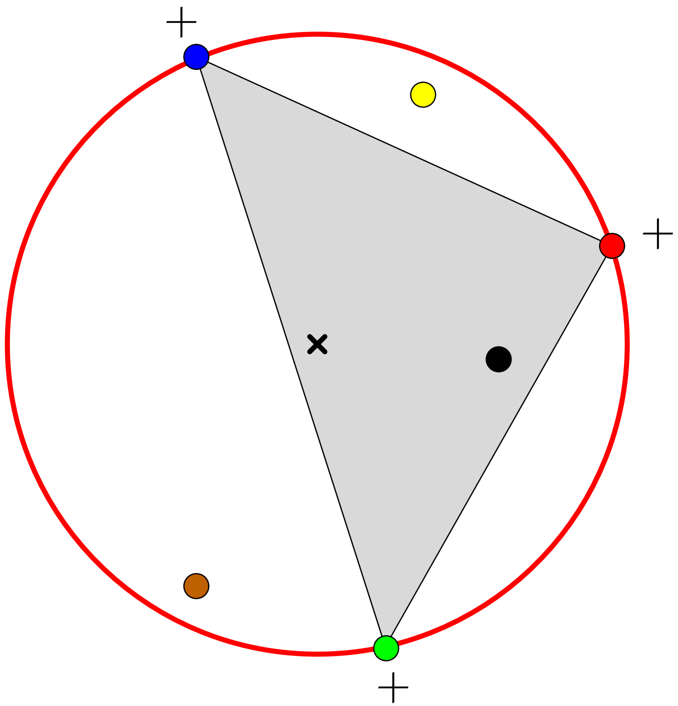
Our Example Again

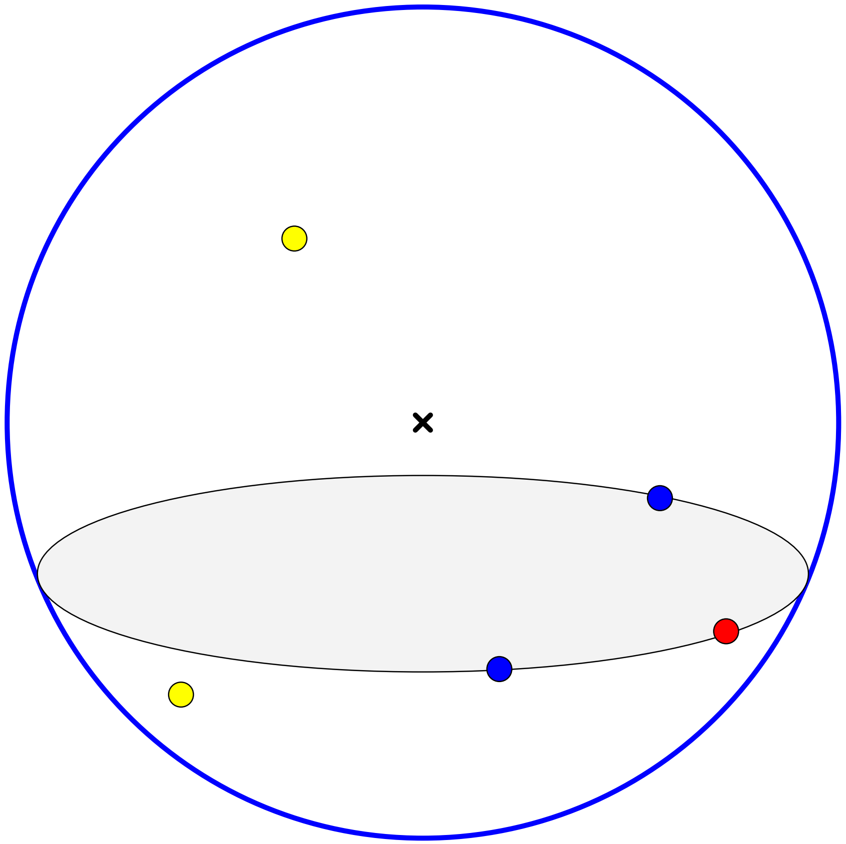


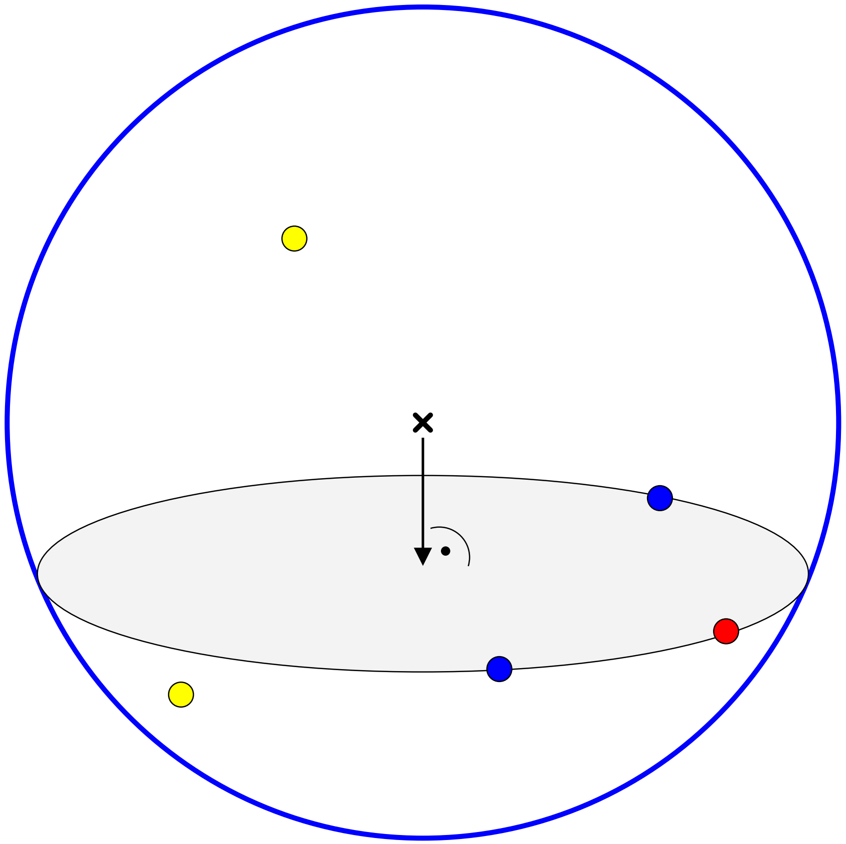
Our Example Again

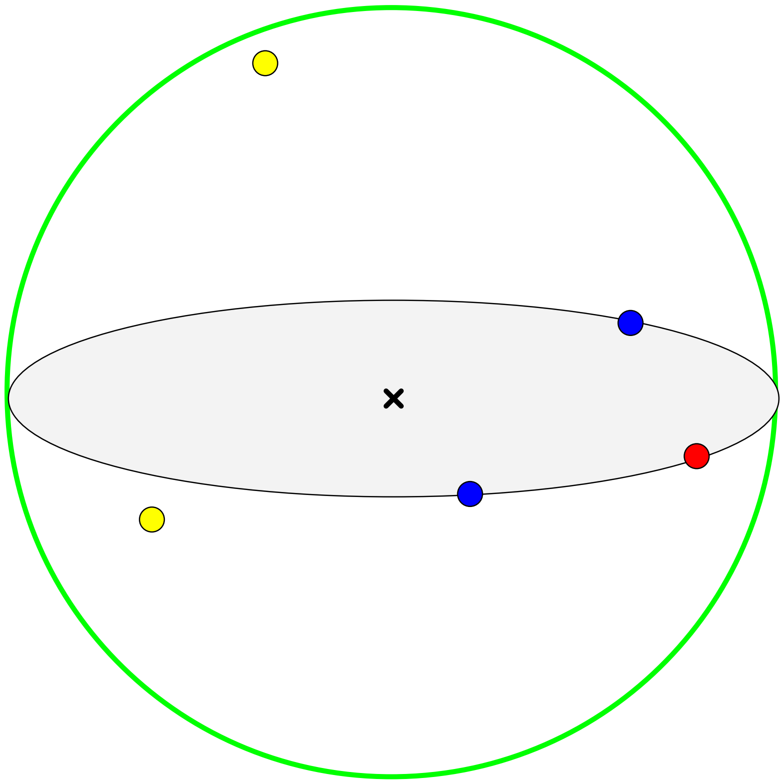


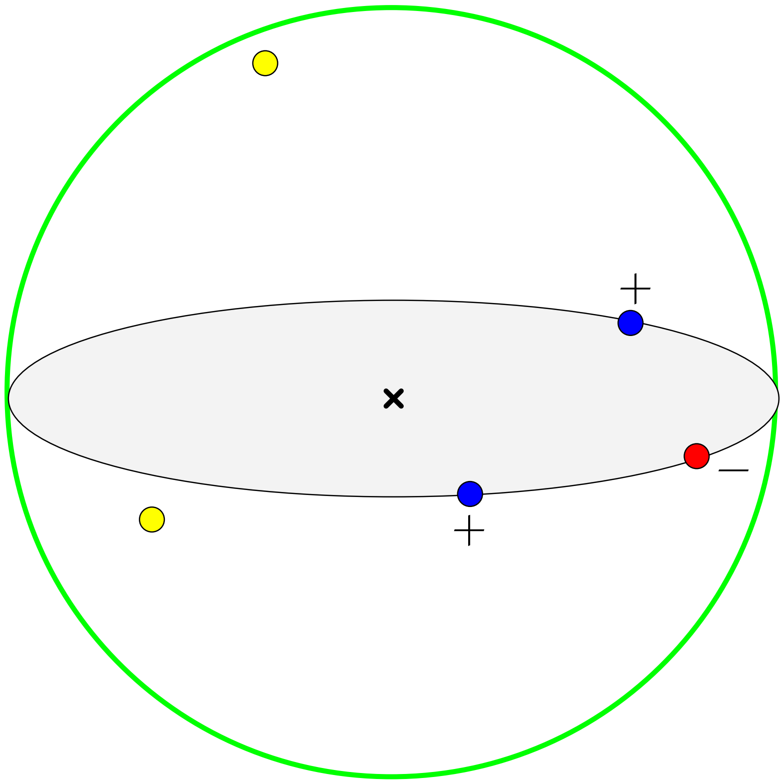
Our Example Again

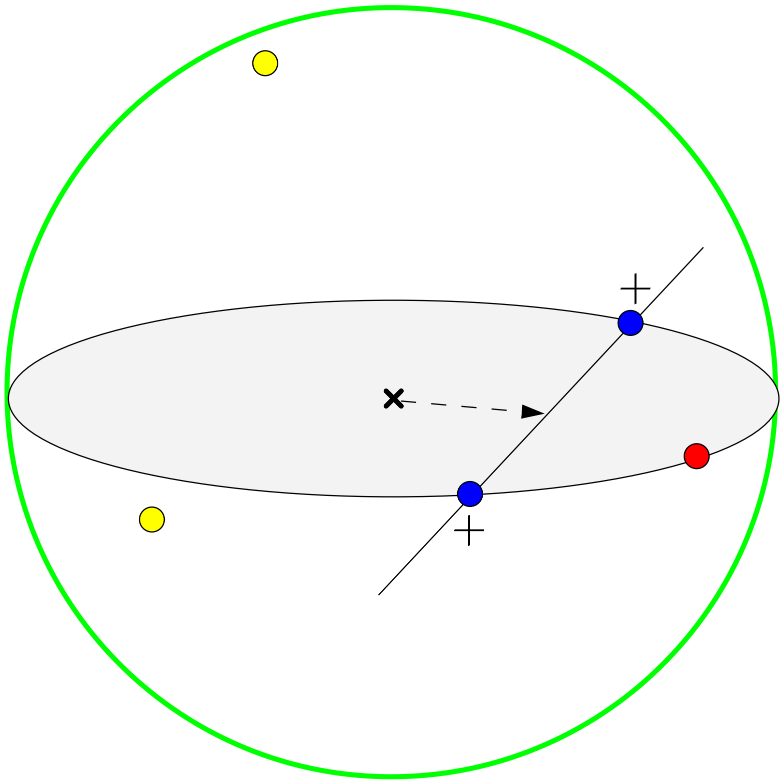












Correctness & Termination

Correctness “clear” from invariants.

Correctness & Termination

Correctness "clear" from invariants.

while $c \notin \text{conv}(T)$ **do**

  [Invariant: $B(c, T) \supset S$, $\partial B(c, T) \supset T$, and T affinely independent]

  if $c \in \text{aff}(T)$ **then** drop T-point with negative coefficient in aff. rep. of c;

  [Invariant: $c \notin \text{aff}(T)$]

  move c towards $\text{aff}(T)$,

  stop when boundary hits new point $q \in S$ or c reaches $\text{aff}(T)$;

  if point stopped us **then** $T := T \cup \{q\}$;

end while;

Correctness & Termination

Correctness “clear” from invariants.   Termination more complicated.

Correctness & Termination

Correctness "clear" from invariants. Termination more complicated.

Proposition. *In the non-degenerate case (no affinely dependent subset* $\mathrm{T} \subseteq S$ *lies on a sphere) the algorithm terminates.*

Correctness & Termination

Correctness "clear" from invariants.   Termination more complicated.

Proposition. *In the non-degenerate case (no affinely dependent subset* $T \subseteq S$ *lies on a sphere) the algorithm terminates.*

Proof:

- Negative-coefficient rule prevents immediate re-insertion after drop.
- Radius decreases after dropping step.
- At least 1 out of d consecutive iterations performs a drop.
- Set of all possible balls $B(c, T)$ preceding drops is finite.

How to Prevent Cycling

In degenerate cases cycling may occur, i.e., the center c doesn't move but only support set T changes — forever.

How to Prevent Cycling

In degenerate cases cycling may occur, i.e., the center c doesn't move but only support set T changes — forever.

Solution: pivot rule, similar to Bland's rule for simplex algorithm.

Index the point set S in arbitrary order.

When dropping a point with negative coefficient, pick the one with smallest index.

When movement stopped by several points, also pick the one with smallest index.

How to Prevent Cycling

In degenerate cases cycling may occur, i.e., the center c doesn't move but only support set T changes — forever.

Solution: pivot rule, similar to Bland's rule for simplex algorithm.

> Index the point set S in arbitrary order.
>
> When dropping a point with negative coefficient, pick the one with smallest index.
>
> When movement stopped by several points, also pick the one with smallest index.

Theorem. *Using "Bland's rule" our algorithm terminates.*

Technical Details

Data structure for support set T needed that allows requests

- compute orthogonal projection onto $\mathrm{aff}(T)$ (for walking),
- compute affine coefficients of point $p \in \mathrm{aff}(T)$ (for dropping)

Technical Details

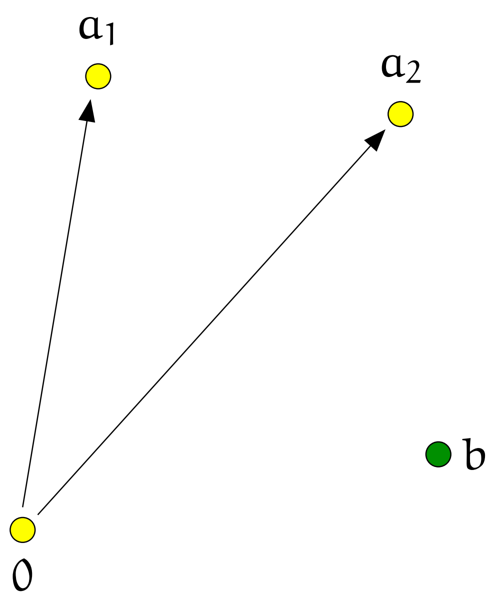
Data structure for support set T needed that allows requests

- compute orthogonal projection onto $\mathrm{aff}(T)$ (for walking),
- compute affine coefficients of point $p \in \mathrm{aff}(T)$ (for dropping)

and updates

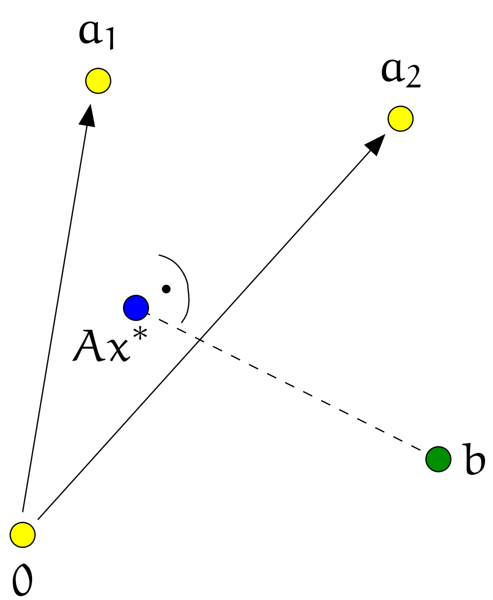
- insert point into T and
- delete point from T.

Technical Details



Let $A := \begin{bmatrix} a_1 & a_2 & \cdots & a_r \end{bmatrix}$ (homogenized).

Technical Details



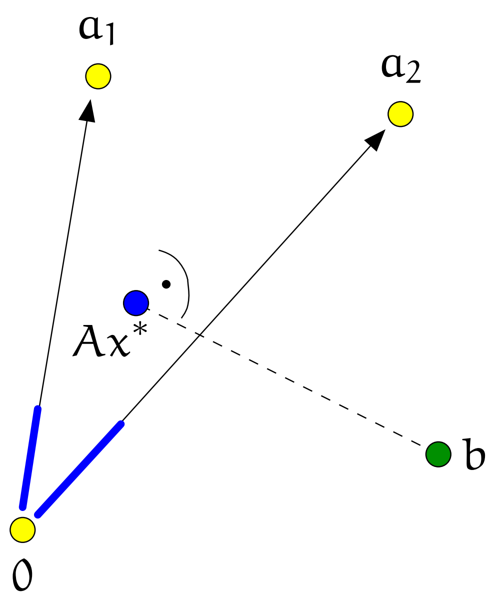
Let $A := \begin{bmatrix} a_1 & a_2 & \cdots & a_r \end{bmatrix}$ (homogenized).

Let $x^* \in \langle a_1, a_2, \ldots, a_k \rangle$ minimize the risidual

$$\|Ax - b\|_2,$$

then Ax^* is the orthogonal projection of b onto $\langle a_1, a_2, \ldots, a_k \rangle$.

Technical Details



Let $A := \begin{bmatrix} a_1 & a_2 & \cdots & a_r \end{bmatrix}$ (homogenized).

Let $x^* \in \langle a_1, a_2, \ldots, a_k \rangle$ minimize the risidual

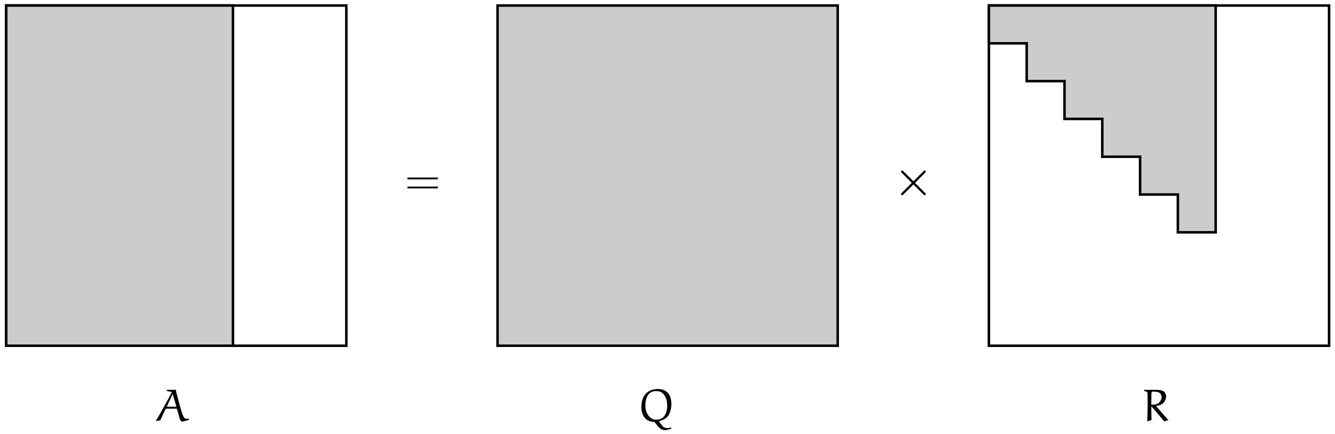
$$||Ax - b||_2,$$

then Ax^* is the orthogonal projection of b onto $\langle a_1, a_2, \ldots, a_k \rangle$.

If $b \in \langle a_1, a_2, \ldots, a_k \rangle$ then the coefficients of x^* simply are the coefficients of b.

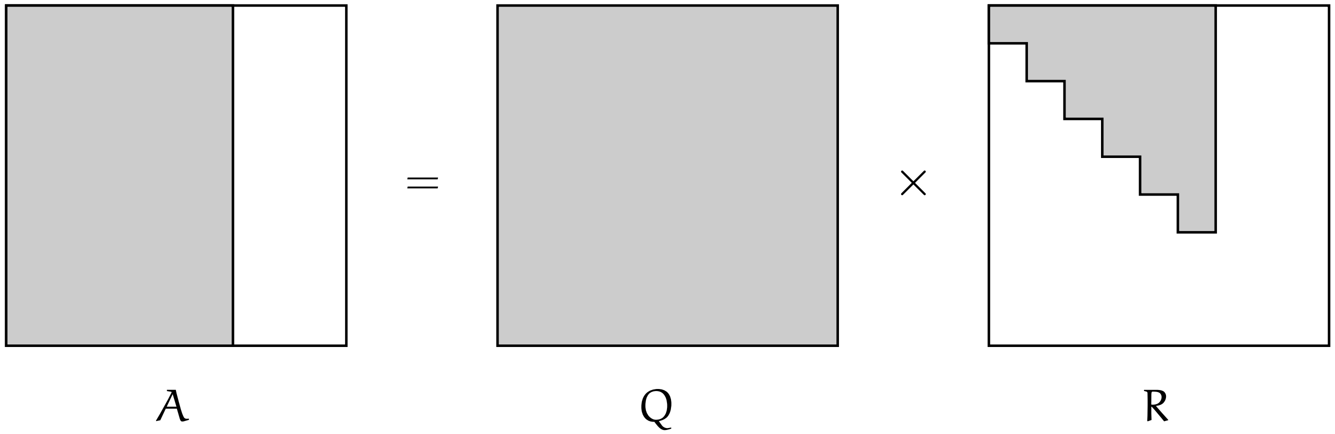
Technical Details

We compute the x^* that minimizes $\|Ax - b\|$ with QR-decomposition



Technical Details

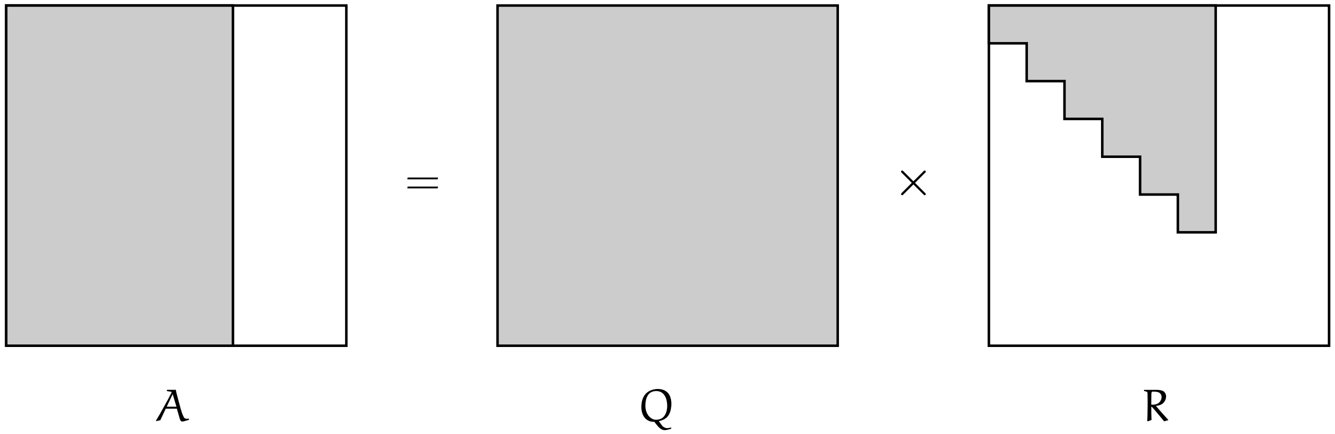
We compute the x^* that minimizes $\|Ax - b\|$ with QR-decomposition



"Solve" $Ax = b$ via $QRx = b \iff Rx = Q^T b$.

Technical Details

We compute the x^* that minimizes $\|Ax - b\|$ with QR-decomposition

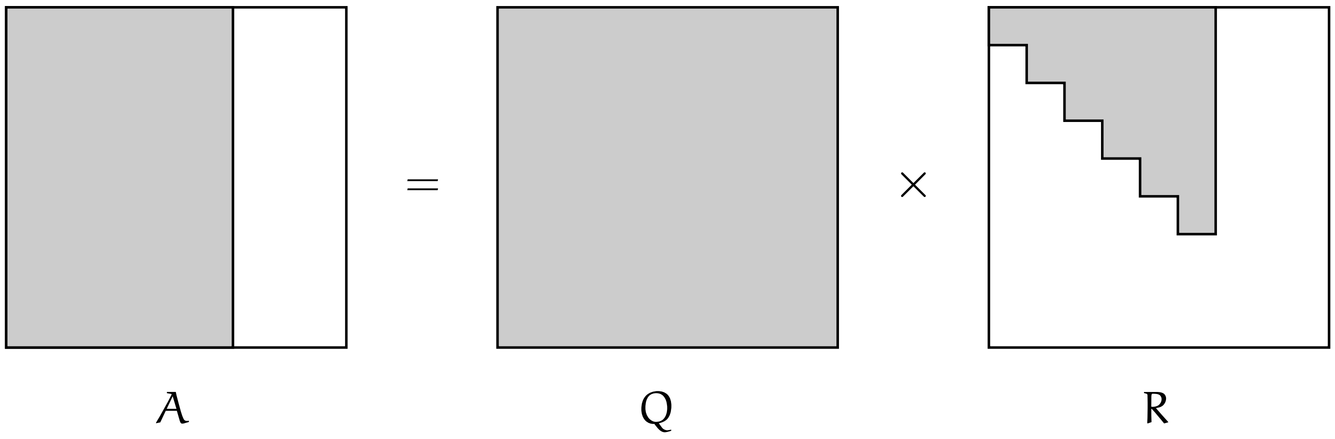


"Solve" $Ax = b$ via $QRx = b \iff Rx = Q^T b$.

Let $y :\approx Q^T b$ with last entries zeroed
and then solve $Rx = y$ via back substitution.

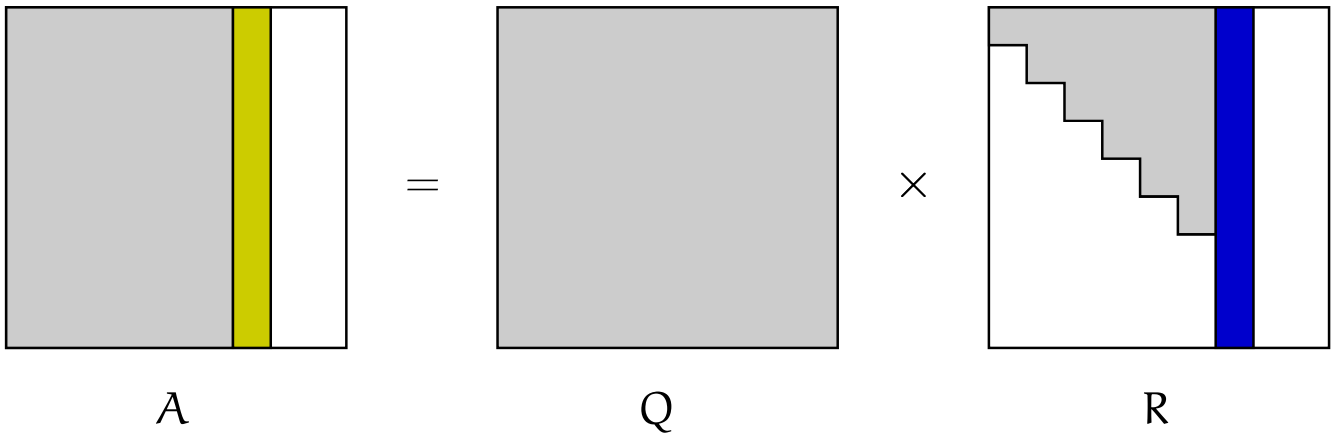
Technical Details

Update QR-decomposition via Givens rotations.



Technical Details

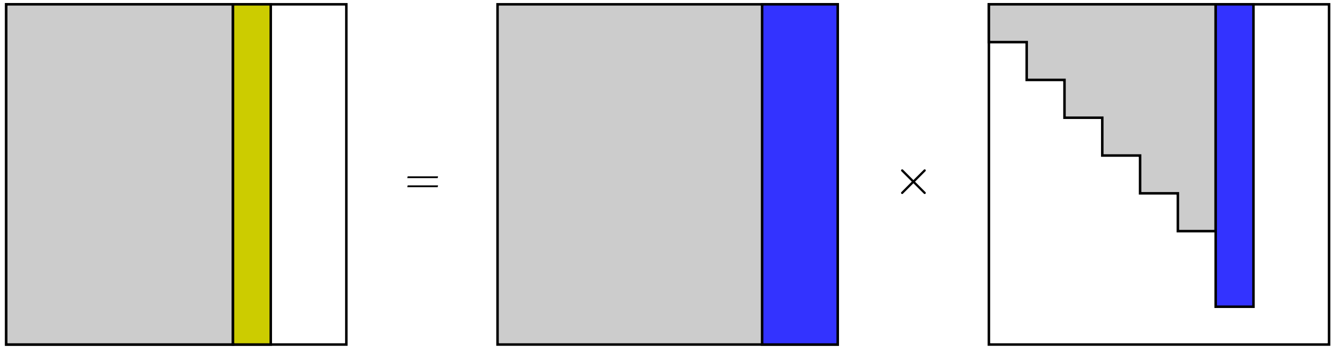
Update QR-decomposition via Givens rotations.



- add new column a to A (and rotated column $Q^T a$ to R)

Technical Details

Update QR-decomposition via Givens rotations.

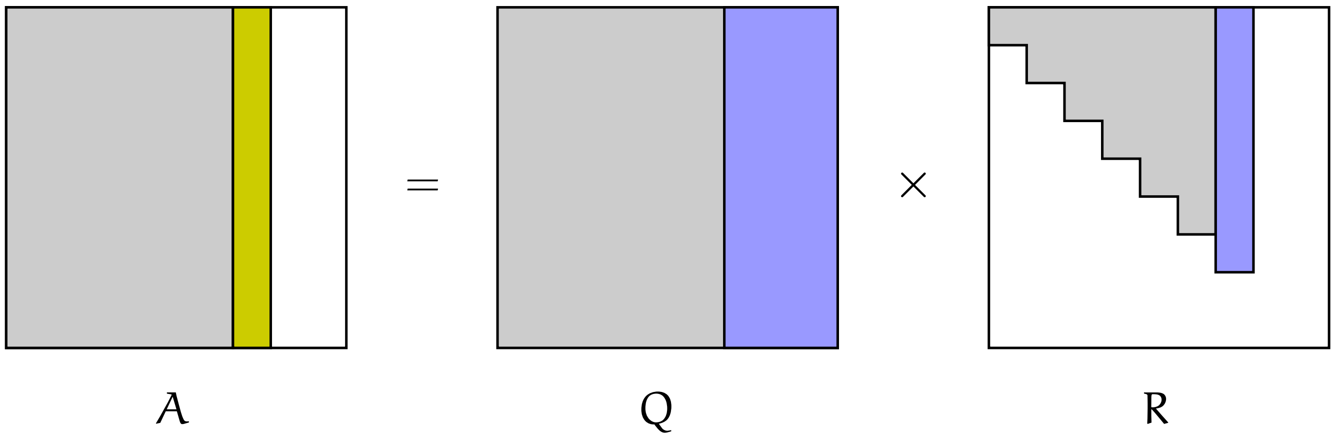


A $\qquad$ Q $\qquad$ R

- add new column a to A (and rotated column $Q^T a$ to R)
- rotate rows of R and colums of Q to reduce R to triangular shape

Technical Details

Update QR-decomposition via Givens rotations.



- add new column a to A (and rotated column $Q^T a$ to R)
- rotate rows of R and colums of Q to reduce R to triangular shape

Our Implementation

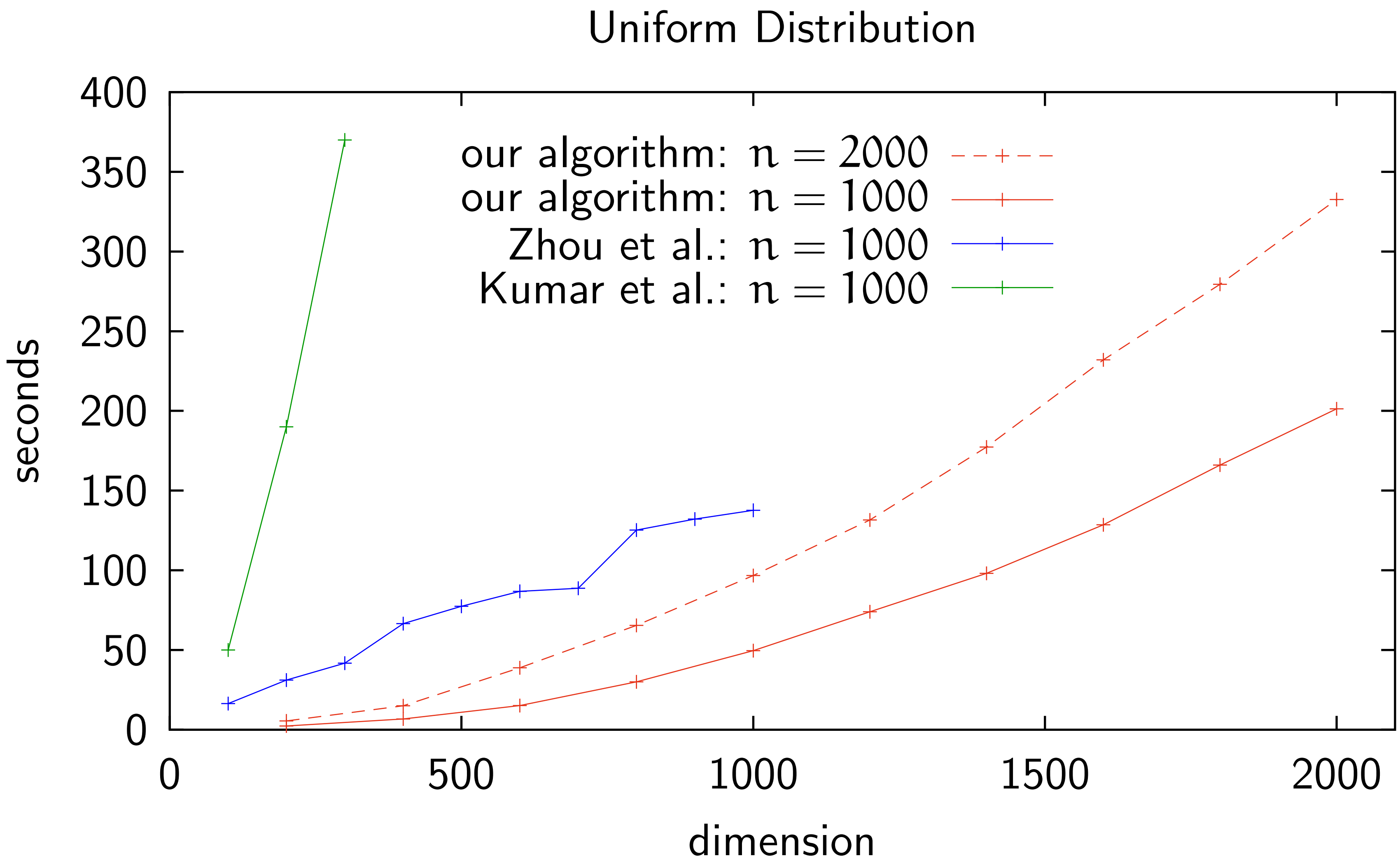
- single iteration in $\mathcal{O}(nd)$ time

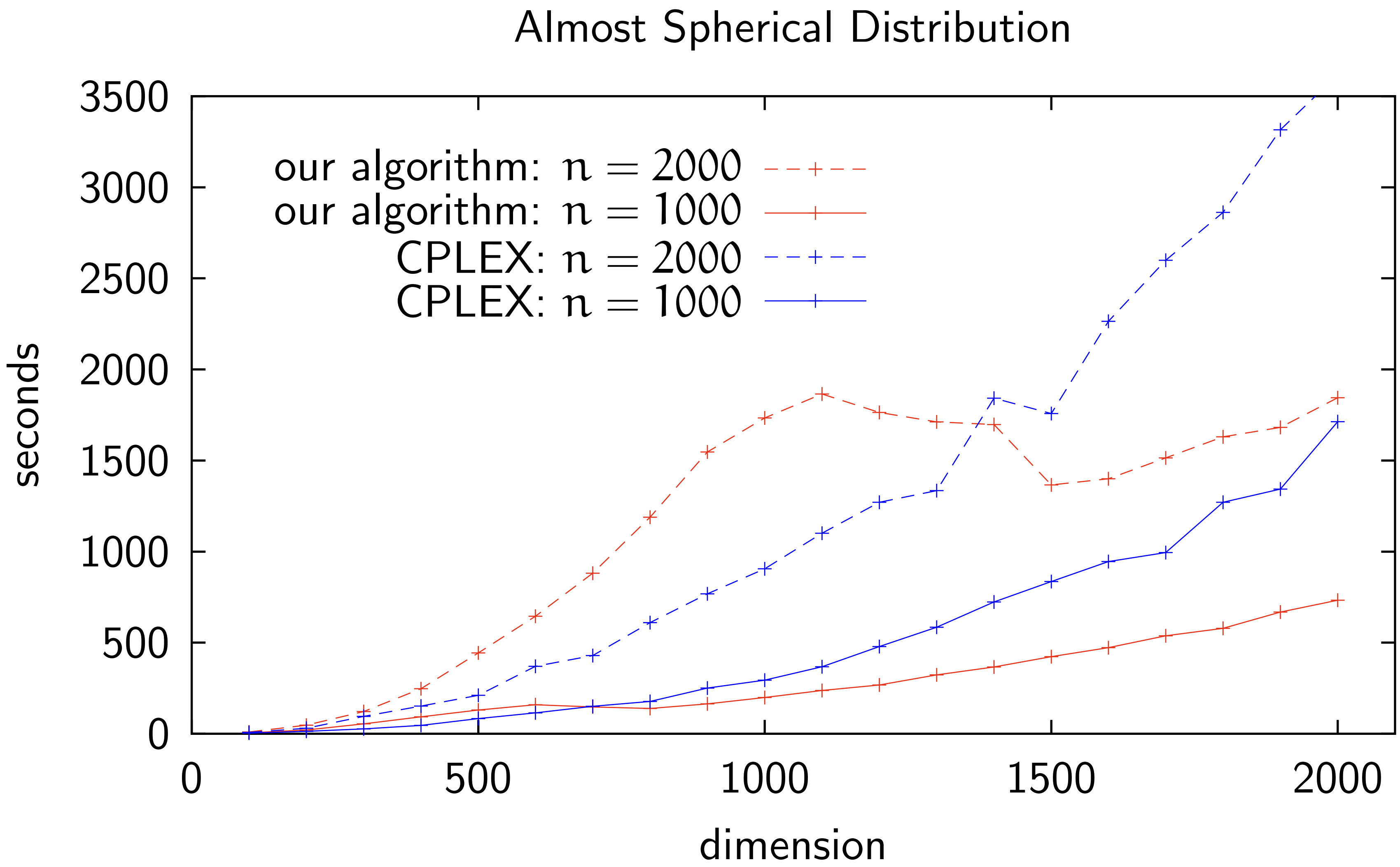
Our Implementation

- single iteration in $\mathcal{O}(nd)$ time
- C++ floating-point
- Bland's rule replaced by numerically more stable heuristic

Our Implementation

- single iteration in $\mathcal{O}(nd)$ time
- C++ floating-point
- Bland's rule replaced by numerically more stable heuristic
- QR decomposition numerically very stable
- very accurate results, about $1{,}000$ times machine precision


Uniform Distribution
our algorithm: $n = 2000$
our algorithm: $n = 1000$
Zhou et al.: $n = 1000$
Kumar et al.: $n = 1000$
seconds
dimension
0
50
100
150
200
250
300
350
400
0
500
1000
1500
2000



Almost Spherical Distribution
seconds
dimension
our algorithm: n = 2000
our algorithm: n = 1000
CPLEX: n = 2000
CPLEX: n = 1000
0
500
1000
1500
2000
2500
3000
3500
0
500
1000
1500
2000