

CS498: Algorithmic Engineering

Lecture 8: Row Generation (LP), Minimum Cut, and The Ellipsoid Method

Elfarouk Harb

University of Illinois Urbana-Champaign

Week 05 – 02/12/2026

Outline

- 1 Quick Note from Last Lecture
- 2 Min $s-t$ Cut Warm-up
- 3 Row Generation for Linear Programs
- 4 Case Study (LP): Minimum $s-t$ Cut

Practical MINLP in Gurobi

Since version 9.0, Gurobi automates Spatial B&B. You do not need to implement McCormick envelopes manually.

The “NonConvex” Parameter

By default, Gurobi assumes quadratic constraints are **convex** (PSD). If you add $z = x \cdot y$ or $y = \sin(x)$, it will throw an error. You must explicitly enable the global solver.

Engineering Note: Just like Big- M , Spatial B&B relies heavily on **variable bounds** (L_x, U_x) to build tight envelopes. **Always bound your continuous variables in MINLP!**

Practical MINLP in Gurobi

```
m = gp.Model("bilinear_example")
x = m.addVar(lb=0, ub=10, name="x") # Bounds are CRITICAL for envelopes!
y = m.addVar(lb=0, ub=10, name="y")
z = m.addVar(name="z")

# 1. Add the bilinear constraint directly
#     Gurobi detects this is non-convex
m.addConstr(z == x * y)

# 2. REQUIRED: Enable non-convex handling
#     0 = error if non-convex (default)
#     2 = translate to McCormick & use Spatial B&B
m.setParam("NonConvex", 2)

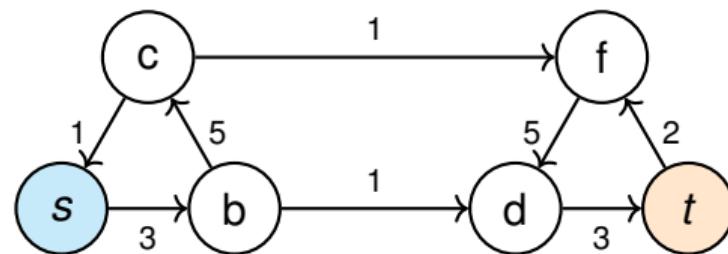
m.optimize()
```

- 1 Quick Note from Last Lecture
- 2 Min $s-t$ Cut Warm-up
- 3 Row Generation for Linear Programs
- 4 Case Study (LP): Minimum $s-t$ Cut

Separating s and t

Story: Given two special vertices s and t in a **directed** graph, we want to disconnect them as cheaply as possible.

Question: What is the minimum total weight of edges we must remove so that there is no path from s to t ?



Intuition: cutting the weak links disconnects s from t .

Definition: $s-t$ Cut

Let $G = (V, E)$ be a **directed** graph with capacities $w_{uv} \geq 0$.

Natural Definition: A cut is a set of edges whose removal disconnects s from t .

Mathematical Definition (Partition): An $s-t$ cut is a partition $(S, V \setminus S)$ such that $s \in S$ and $t \notin S$. The capacity of the cut is the weight of edges **leaving** S :

$$w(\delta^+(S)) = \sum_{\substack{(u,v) \in E \\ u \in S, v \notin S}} w_{uv}$$

Why this definition?

Why do we focus on partitions? Because any **minimal** set of edges that separates s from t forms a cut $\delta^+(S)$, where S is the set of nodes reachable from s .

The Max-Flow Min-Cut Theorem [CS473]

How do we compute the Minimum Cut?

Theorem (Max-Flow Min-Cut)

The maximum amount of flow possible from s to t is equal to the capacity of the minimum s - t cut.

Why is this useful for us?

- **Flow** is naturally a Linear Programming (LP) problem (continuous variables).
- **Cut** looks like a discrete/integer optimization problem (picking a set S).

This theorem allows us to solve the discrete Cut problem efficiently using Linear Programming and Max-Flow Algorithms!

A Note on LP Formulations

There are different ways to write an LP for Minimum $s - t$ Cut.

- ① **Edge-based Flow Formulation (Standard):** Variables f_{uv} on edges.
Constraints involve flow conservation.
- ② **Path-based (What we will do today):** This is actually the **Dual** of the path-based flow formulation.

Caution

The formulation we show next is useful for understanding row-generation, but it is **not** the standard textbook flow formulation you might see in other contexts. For standard flow algorithms, see CS473.

Example: What is the Min $s-t$ Cut Here?

Cutting inside either dense cluster costs at least 4 or 5.
But cutting the two crossing edges costs:

$$1 + 1 = 2.$$

So the minimum $s-t$ cut is defined by the set:

$$S = \{s, b, c\}$$

And the cut value is $w(\delta^+(S)) = 2$.

In Python: NetworkX Minimum $s-t$ Cut

```
import networkx as nx

G = nx.DiGraph() # Explicitly Directed

# Add edges with capacities
G.add_edge("s", "b", capacity=3)
G.add_edge("b", "c", capacity=2)
G.add_edge("c", "s", capacity=2)

G.add_edge("d", "t", capacity=3)
G.add_edge("t", "f", capacity=2)
G.add_edge("f", "d", capacity=2)

G.add_edge("b", "d", capacity=1)
G.add_edge("c", "f", capacity=1)

# NetworkX uses Max-Flow Min-Cut theorem internally
cut_value, partition = nx.minimum_cut(G, "s", "t")
S, T = partition

print("min st cut value =", cut_value)
print("S =", S)
```

- 1 Quick Note from Last Lecture
- 2 Min $s-t$ Cut Warm-up
- 3 Row Generation for Linear Programs
- 4 Case Study (LP): Minimum $s-t$ Cut

Row Generation: When Constraints Are Too Many to Write

Suppose your LP has:

- 10,000 variables (fine),
- but 10^{100} constraints (not fine).

Key idea

Most constraints are *not* binding at optimum.

So: start small, then add only what you need.

The Abstraction: Full LP vs Restricted LP

Let the true LP be L :

$$\min c^\top x \quad \text{s.t. } a_k^\top x \leq b_k \quad \forall k \in \mathcal{K}, \quad x \in X$$

\mathcal{K} is huge (maybe exponential).

We solve a restricted LP L' with only $\mathcal{K}' \subset \mathcal{K}$.

Two outcomes:

- x^* violates none $\Rightarrow x^*$ is optimal for L , we are done!
- x^* violates some constraint in L \Rightarrow add it to L' .

Sometimes 1 Constraint is Already Enough

Toy LP with many constraints (conceptually):

$$\min x \quad \text{s.t. } x \geq 1, x \geq 2, x \geq 3, \dots, x \geq 10^{100}.$$

Here, $\mathcal{K} = \{1, 2, \dots, 10^{100}\}$ and each constraint is $x \geq k$.

Restricted LP L' with only the “last” constraint

Take $\mathcal{K}' = \{10^{100}\}$:

$$\min x \quad \text{s.t. } x \geq 10^{100}.$$

Solve L' : $x^* = 10^{100}$.

Now check feasibility in the full LP:

$$x^* = 10^{100} \geq k \quad \forall k \in \{1, \dots, 10^{100}\}.$$

Outcome (we are done)

No violated constraints $\Rightarrow x^*$ is optimal for the full LP L .

Sometimes 1 Constraint is Not Enough

Restricted LP L' with only the first constraint

Take $\mathcal{K}' = \{1\}$:

$$\min x \quad \text{s.t. } x \geq 1.$$

Solve L' : $x^* = 1$.

Now check feasibility in the full LP:

$$x^* = 1 \not\geq k \quad \forall k \in \{1, \dots, 10^{100}\}.$$

Outcome

Some violated constraints, for example $x \geq 2$.

Separation Oracle (Definition)

A **Separation oracle** for L is an **Algorithm** such that:

Input / Output

Input: candidate solution x^* .

Output:

- either: “no violated constraints” (so x^* is feasible),
- or: return a violated inequality $a^\top x \leq b$.

Row generation = repeatedly call the oracle and add returned rows.

Row Generation Algorithm (High Level)

- ① Initialize \mathcal{K}' with a small set of constraints.
- ② Solve restricted LP $L'(\mathcal{K}')$.
- ③ Run oracle on x^* :
 - ▶ If violated constraint found: add it to \mathcal{K}' and repeat.
 - ▶ Else: stop. Current x^* is optimal for full LP.

What We Need for Row Generation to Be Practical

- Restricted LP must be solvable fast (yes).
- Oracle must be fast (poly-time), not just enumerating \mathcal{K} !
- The number of iterations in practice should be reasonable.

So the lecture question becomes:

Can this actually happen? Can we have a huge constraint set, but a fast separation oracle?

- 1 Quick Note from Last Lecture
- 2 Min $s-t$ Cut Warm-up
- 3 Row Generation for Linear Programs
- 4 Case Study (LP): Minimum $s-t$ Cut

Back to Minimum $s-t$ Cut

At the beginning of the lecture, we saw:

Problem

Given a graph with capacities $w_e \geq 0$, find the cheapest set of edges whose removal disconnects s from t .

Now we model this as an optimization problem.

Step 1: What Are Our Decisions?

For every edge $e \in E$, define:

$$y_e = \begin{cases} 1 & \text{if we cut edge } e \\ 0 & \text{otherwise} \end{cases}$$

So y_e is a binary decision variable.

If we cut edge e , we pay cost w_e .

Step 2: Objective Function

We want to minimize total cutting cost:

$$\min \sum_{e \in E} w_e y_e$$

So far, this just says:

- Choose some edges to cut,
- Pay their total capacity.

But we have not yet enforced that s and t are disconnected.

Step 3: What Does It Mean to Disconnect s and t ?

s and t are disconnected if:

Separation

Every path from s to t contains at least one cut edge.

In other words:

$$\text{For every } s-t \text{ path } P, \quad \sum_{e \in P} y_e \geq 1.$$

Why This Constraint is Correct

Consider any path P from s to t .

If:

$$\sum_{e \in P} y_e = 0,$$

then none of the edges on P were cut.

That means the path still exists.

So to eliminate the path, we must enforce:

$$\sum_{e \in P} y_e \geq 1.$$

Full Integer Programming Model For Min $s - t$ Cut.

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e y_e \\ \text{s.t.} \quad & \sum_{e \in P} y_e \geq 1 \quad \forall s-t \text{ paths } P \\ & y_e \in \{0, 1\} \end{aligned}$$

This is a correct formulation of minimum $s-t$ cut.

Now Look Carefully at the Constraints

We have:

$$\sum_{e \in P} y_e \geq 1 \quad \forall s-t \text{ paths } P.$$

How many $s-t$ paths are there?

Problem

Potentially exponentially many.

Relax Integrality: Full Linear Programming Model

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e y_e \\ \text{s.t.} \quad & \sum_{e \in P} y_e \geq 1 \quad \forall s-t \text{ paths } P \\ & 0 \leq y_e \leq 1 \end{aligned}$$

Proving Polytope Integrality

Let $G = (V, E)$ be a directed graph with terminals s, t . Consider a feasible fractional solution $y \in \mathbb{R}_{\geq 0}^E$ to the path LP:

$$\sum_{e \in P} y_e \geq 1 \quad \forall s-t \text{ paths } P.$$

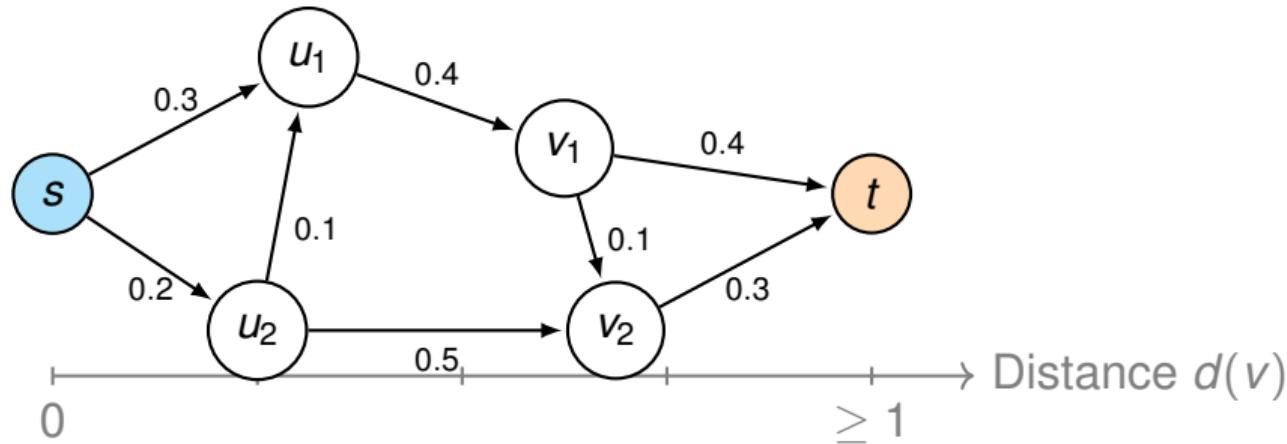
Define shortest-path distances from s using edge lengths y_e :

$$d(v) := \min_{P: s \rightsquigarrow v} \sum_{e \in P} y_e.$$

Then:

$$d(s) = 0, \quad d(t) \geq 1.$$

Visualizing the Distance Metric



We arrange the nodes on the horizontal axis according to their shortest path distance $d(v)$ from s using the fractional weights y_e .

The Sweeping Cut

For each $r \in [0, 1]$, define:

$$S(r) := \{v \in V : d(v) \leq r\}.$$

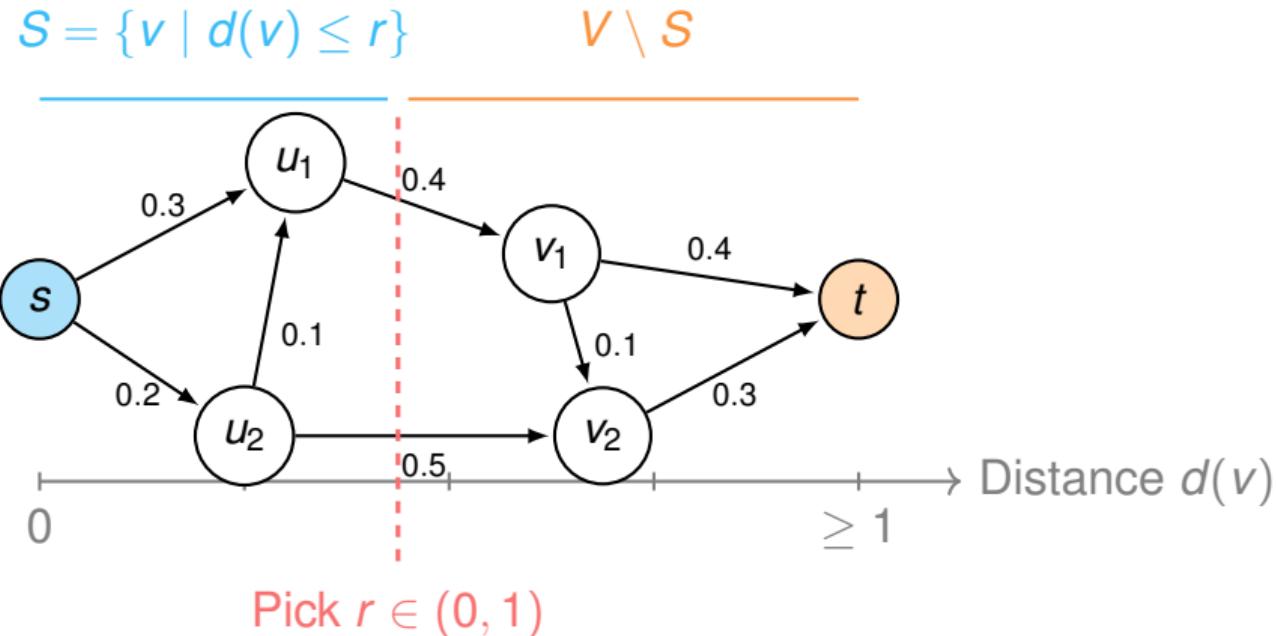
Define the cut:

$$C(r) := \delta^+(S(r)) = \{(u, v) \in E : d(u) \leq r < d(v)\}.$$

This is an $s-t$ cut since:

$$d(s) = 0 \leq r, \quad d(t) \geq 1 > r.$$

The Random Threshold Cut



Algorithm: Pick a radius $r \in (0, 1)$. Say $r = 0.35$

Cut all edges (u, v) where u is "inside" ($d(u) \leq r$) and v is "outside" ($d(v) > r$).

Switching the Perspective

We study the integrated cut cost:

$$\int_0^1 \text{cost}(C(r)) dr.$$

The cost of a cut is:

$$\text{cost}(C(r)) = \sum_{e \in C(r)} w_e.$$

Swap summation and integration:

$$\int_0^1 \text{cost}(C(r)) dr = \int_0^1 \sum_{e \in C(r)} w_e dr = \sum_{e=(u,v) \in E} w_e \cdot (\text{length of } r \text{ where } e \text{ is in } C(r)).$$

An edge (u, v) is cut exactly when $d(u) \leq r < d(v)$. The length of this interval is $\max\{0, d(v) - d(u)\}$.

Triangle Inequality

Shortest-path distances satisfy (Triangle inequality):

$$d(v) \leq d(u) + y_{uv}.$$

Thus:

$$\max\{0, d(v) - d(u)\} \leq \max\{0, y_{uv}\} = y_{uv}$$

Summing over all edges:

$$\sum_{e \in E} w_e \cdot (\text{length of } r \text{ when } e \text{ is in } C(r)) = \sum_{e=(u,v) \in E} w_e \cdot \max\{0, d(v) - d(u)\} \leq \sum_{e \in E} w_e y_e.$$

This equals the LP cost.

Conclusion: A Good Integral Cut Exists

We have shown:

$$\int_0^1 \text{cost}(C(r)) dr \leq \text{LP Cost.}$$

Therefore by mean value theorem, there exists some $r^* \in [0, 1]$ such that:

$$\text{cost}(C(r^*)) \leq \text{LP Cost.}$$

We proved that any fractional y is an average of integer cuts. A vertex cannot be an average of other things. Therefore, if y is a vertex, it must inherently be an integer cut itself.

Full LP Model For Min $s - t$ Cut.

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e y_e \\ \text{s.t.} \quad & \sum_{e \in P} y_e \geq 1 \quad \forall s-t \text{ paths } P \\ & 0 \leq y_e \leq 1 \end{aligned}$$

This is an integral polytope. Solving it gives us $y_e \in \{0, 1\}$.

Now the Real Question

We have an LP with:

- Exponentially many constraints,
- But only polynomially many variables.

How can we solve it?

Row Generation Framework

We do not list all path constraints.

Instead:

- 1 Start with no path constraints.
- 2 Solve the restricted LP.
- 3 Check if some path constraint is violated.
- 4 If yes, add it.
- 5 Repeat.

Separation Oracle Step

Given current solution y , we want to see if any constraint is violated.

$$\sum_{e \in P} y_e \geq 1 \quad \forall s-t \text{ paths } P$$

That is, find an $s-t$ path P such that

$$\sum_{e \in P} y_e < 1$$

Shortest $s-t$ path problem! Dijkstra!

Define edge lengths $\ell_e := y_e$.

Compute shortest path from s to t :

$$\min_{\forall s-t \text{ paths } P} \sum_{e \in P} y_e.$$

If shortest path length < 1 , that path gives a violated constraint.

Now We Actually Implement Row Generation

We now implement the algorithm:

- ① Build restricted LP (no path constraints).
- ② Solve LP.
- ③ Run shortest path under weights y^* .
- ④ If path length < 1 , add that constraint.
- ⑤ Repeat.

This is real LP row generation.

Step 1: Build the Restricted LP

```
m = gp.Model("min_st_cut_path_LP")

# y_e in [0,1]
y = m.addVars(E, lb=0.0, ub=1.0,
               vtype=GRB.CONTINUOUS, name="y")

# Objective: minimize total cut capacity
m.setObjective(
    gp.quicksum(c[e] * y[e] for e in E),
    GRB.MINIMIZE
)
```

At this point:

- No path constraints yet.
- LP will initially set all $y_e = 0$.

Step 2: Row Generation Loop

```
MAX_ITERS = 100

for it in range(MAX_ITERS):
    m.optimize()
    ystar = {e: y[e].X for e in E}

    # Build graph with edge weights = y*
    G = nx.DiGraph()
    for (u,v) in E:
        G.add_edge(u, v, weight=ystar[(u,v)])

    dist, path = nx.single_source_dijkstra(G, s, t)
```

Now we check whether the shortest path violates a constraint.

Step 3: Check Violation

```
if dist >= 1:  
    print("No violated path constraints.")  
    break
```

If shortest path length ≥ 1 :

Termination

All path constraints are satisfied.

We are done.

Step 4: Add Violated Constraint

```
# Add constraint for violated path
m.addConstr(
    gp.quicksum(
        y[(path[i], path[i+1])]
        for i in range(len(path)-1)
    ) >= 1
)
```

This adds exactly one new row (constraint):

$$\sum_{e \in P} y_e \geq 1.$$

Then we re-solve the LP.

Why This Works

- The LP has exponentially many path constraints.
- We never list them explicitly.
- Separation Oracle reduces to shortest path (poly-time Dijkstra).
- Therefore, row generation solves the LP.

And because min $s-t$ cut LP is integral, the final solution is actually a true cut.

Final Min s-t Code

```
import gurobipy as gp
import networkx as nx
s, t = "s", "t"
E = [("s", "a"), ("s", "b"), ("a", "t"), ("b", "t"), ("a", "b")]
c = {("s", "a"): 2, ("s", "b"): 1, ("a", "t"): 1, ("b", "t"): 3, ("a", "b"): 1}
m = gp.Model("min_st_cut_path_LP")
y = m.addVars(E, lb=0.0, ub=1.0, vtype=gp.GRB.CONTINUOUS, name="y")
m.setObjective(gp.quicksum(c[e] * y[e] for e in E), gp.GRB.MINIMIZE)

for it in range(100):
    m.optimize()
    # Extract current solution
    ystar = {e: y[e].X for e in E}
    # Build graph with edge weights = ystar
    G = nx.DiGraph()
    for (u,v) in E: G.add_edge(u, v, weight=ystar[(u,v)])
    # Shortest path from s to t
    dist, path = nx.single_source_dijkstra(G, s, t)
    # Check violation
    if dist >= 1: break
    m.addConstr(gp.quicksum(y[(path[i], path[i+1])] for i in range(len(path)-1)) >= 1)
#{('s', 'a'): 0.0, ('s', 'b'): 1.0, ('a', 't'): 1.0, ('b', 't'): 0.0, ('a', 'b'): 1.0}
```

Efficiency: Cold Start vs. Hot Start

We are solving a sequence of Linear Programs: $LP_0, LP_1, LP_2 \dots$ where each is slightly larger than the last.

The Naive Approach (Cold Start): If we rebuild the model object from scratch every iteration:

- We throw away the previous optimal basis.
- The solver must start from scratch every time.
- **Cost:** Expensive.

The Smart Approach (Hot Start):

- Keep the `gp.Model` object alive.
- Just add the new constraint to the existing object.
- **Benefit:** Gurobi re-uses the internal state (basis) from the last solve.

Implementation: Where to put the Model?

Bad (Cold Start)

```
# Re-creating the model  
# inside the loop is slow!  
  
for it in range(100):  
  
    m = gp.Model() # <--- BAD  
    y = m.addVars(...)  
  
    # You have to re-add  
    # ALL previous constraints  
    # manually here.  
  
    m.optimize()
```

Good (Hot Start)

```
# Create model ONCE  
m = gp.Model() # <--- GOOD  
y = m.addVars(...)  
  
for it in range(100):  
  
    # Solve existing model  
    m.optimize()  
  
    # ... find violation ...  
  
    # Modifies existing matrix  
    m.addConstr(...)
```

In the **Good** case, ‘m.optimize()’ detects the existing basis and performs a "Warm Start" or "Hot Start" automatically.

Under the Hood: Dual Simplex

Why is hot-starting so effective for Row Generation?

- **Primal Solution:** The old solution y^* is now infeasible (because we just added a constraint that cuts it off).
- **Dual Solution:** The old dual variables are often still feasible (or close to it) for the dual problem.

Algorithm Choice

When we call ‘`m.optimize()`’ after adding a constraint, Gurobi typically switches to the **Dual Simplex** algorithm.

It starts from the previous basis and usually requires only a few pivots to fix the infeasibility, rather than traversing the whole polytope again.

Ellipsoid Method: A Beautiful Theoretical Result

Row generation works extremely fast in practice. In practice we add a very small number of constraints and can reach an optimal solution.

But what about theoretically?

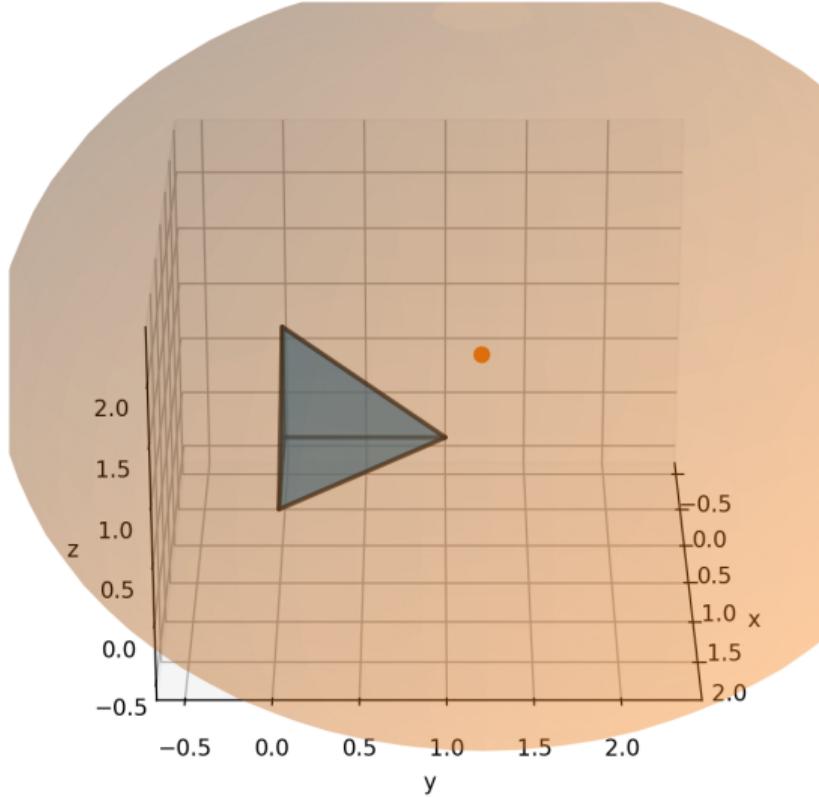
Ellipsoid Method Theorem

If a polytope admits a polynomial-time separation oracle for its constraints, then we can optimize a linear function over it (LP) in polynomial time (ellipsoid method).

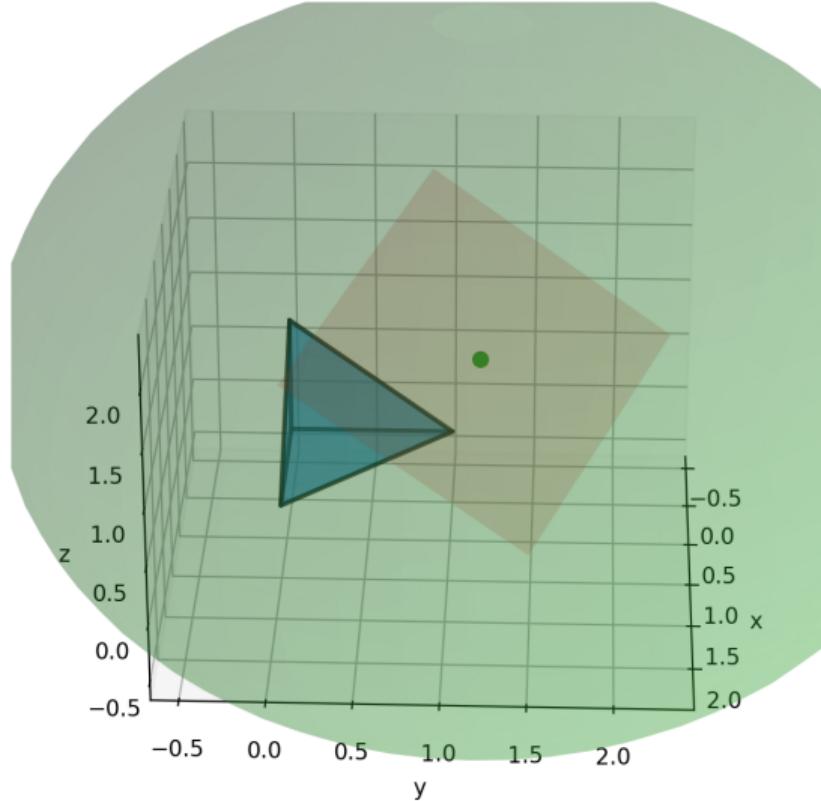
So for LPs:

- exponential constraints are not automatically hopeless,
- if we can separate efficiently.

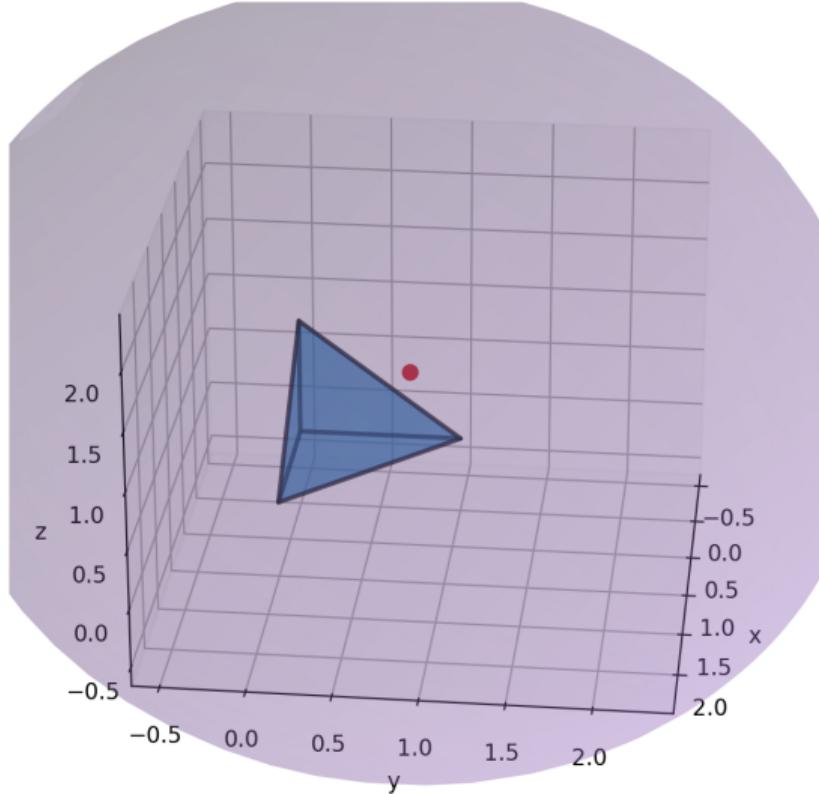
Ellipsoid Method (3D) — iter 0 (show ellipsoid)
infeasible center | constraint cut | max-axis≈2.800



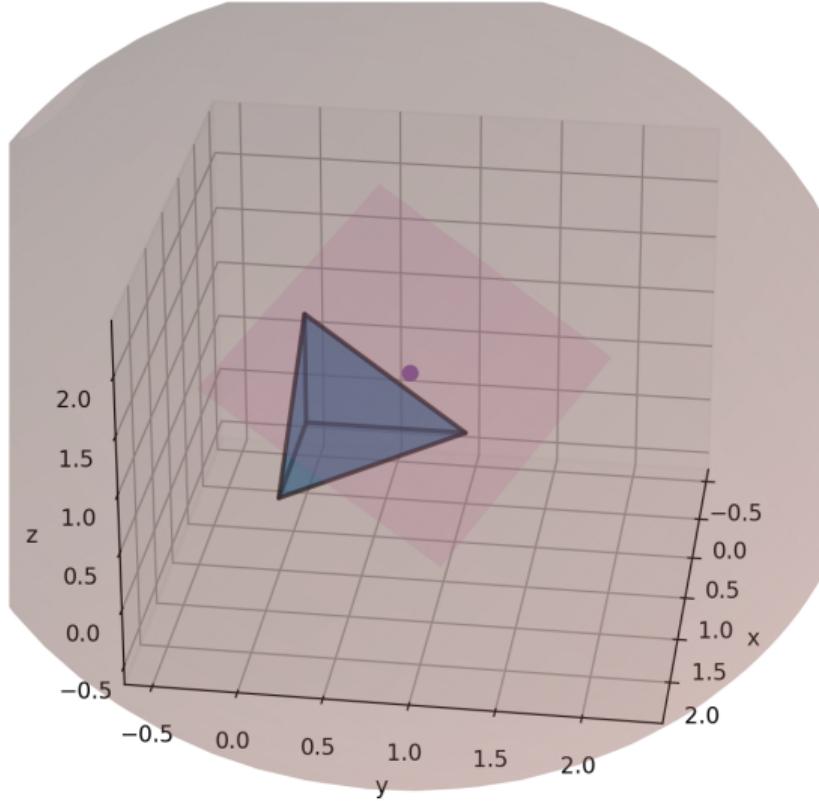
Ellipsoid Method (3D) — iter 0 (show cut plane)
infeasible center | constraint cut | max-axis≈2.800



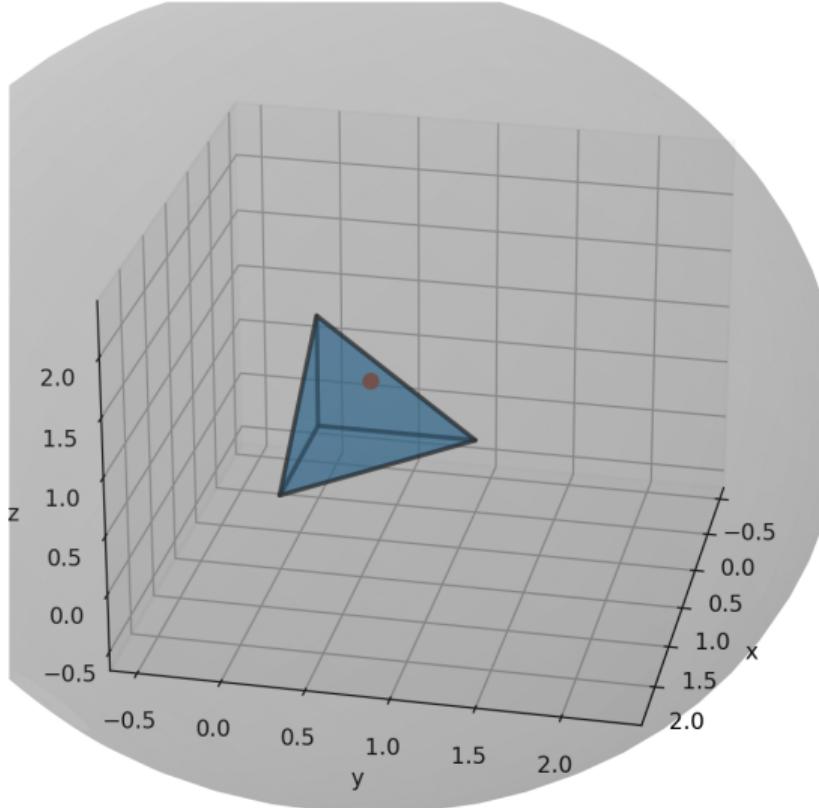
Ellipsoid Method (3D) — iter 1 (show ellipsoid)
infeasible center | constraint cut | max-axis≈2.970



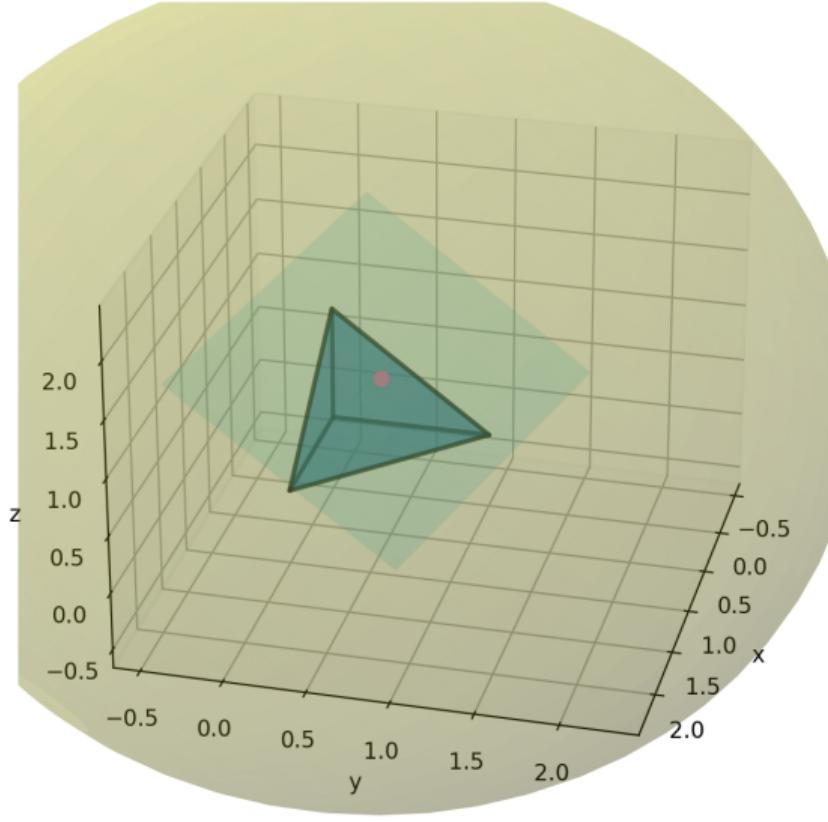
Ellipsoid Method (3D) — iter 1 (show cut plane)
infeasible center | constraint cut | max-axis≈2.970



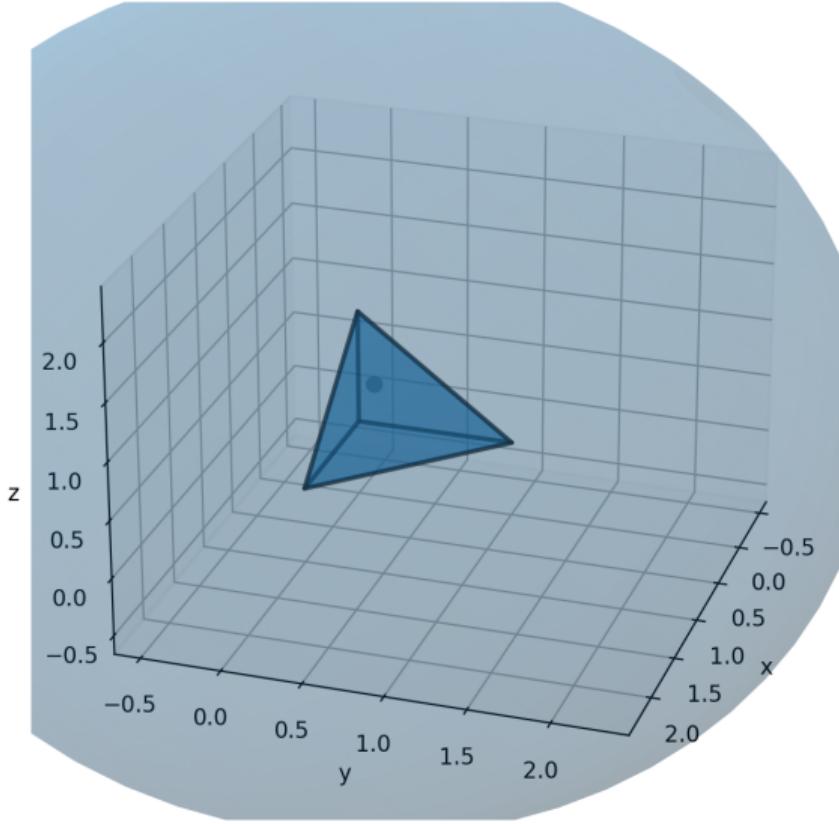
Ellipsoid Method (3D) — iter 2 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 3.150



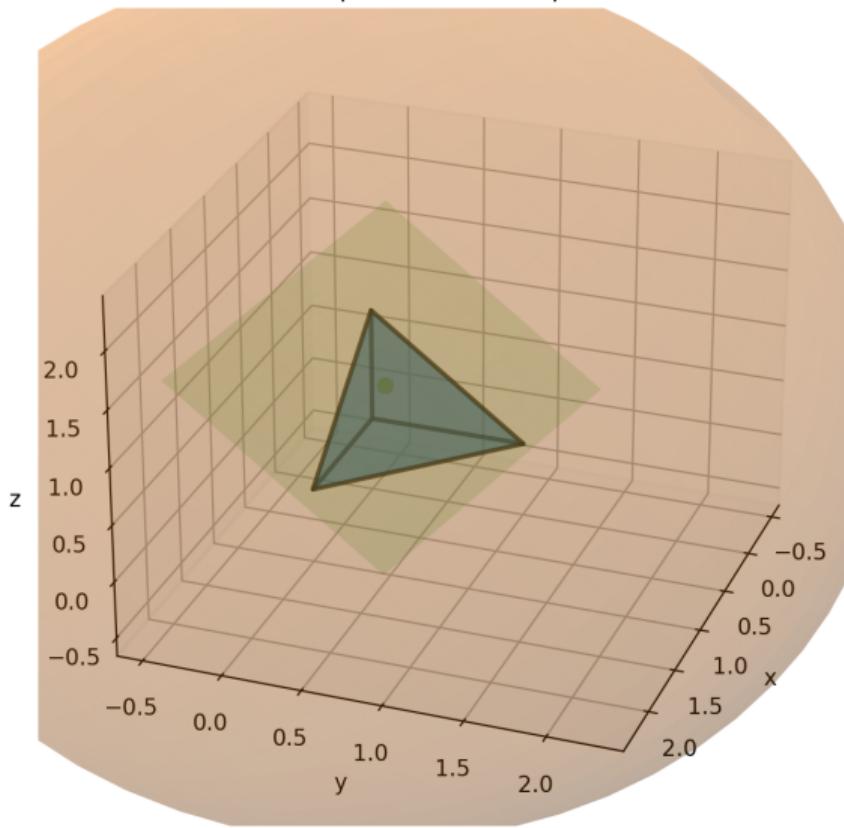
Ellipsoid Method (3D) — iter 2 (show cut plane)
infeasible center | constraint cut | max-axis≈3.150



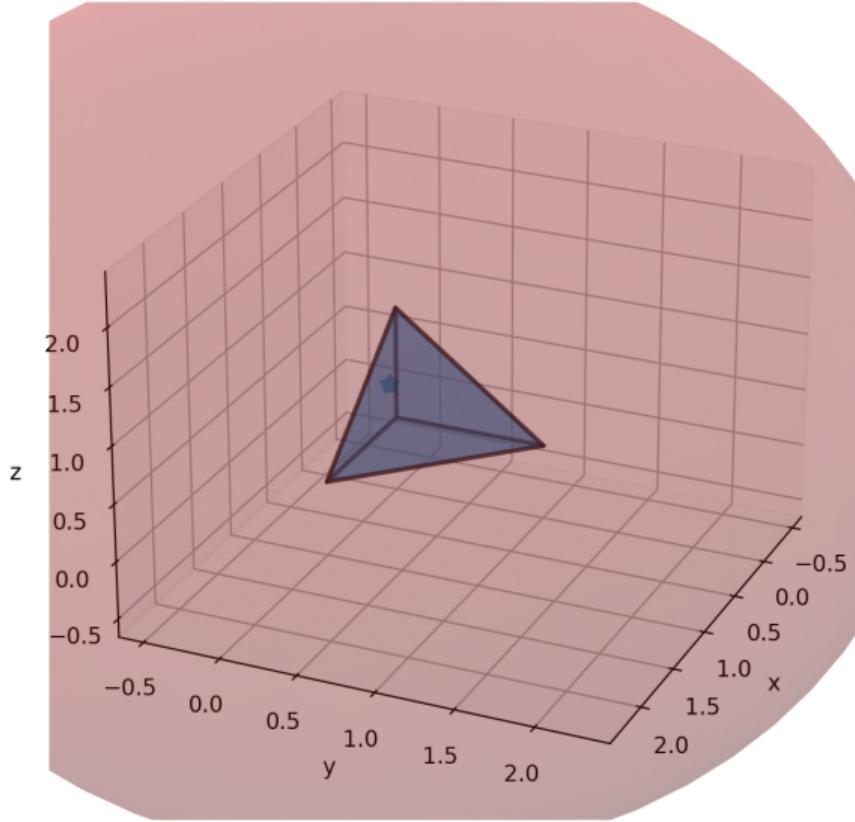
Ellipsoid Method (3D) — iter 3 (show ellipsoid)
infeasible center | constraint cut | max-axis≈3.341



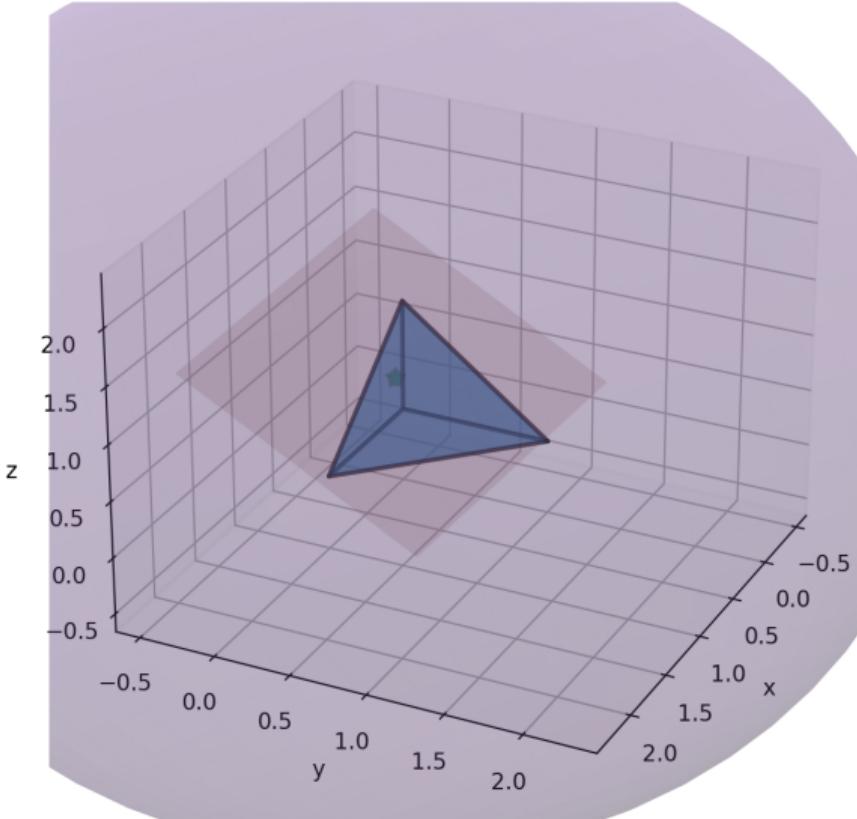
Ellipsoid Method (3D) — iter 3 (show cut plane)
infeasible center | constraint cut | max-axis≈3.341



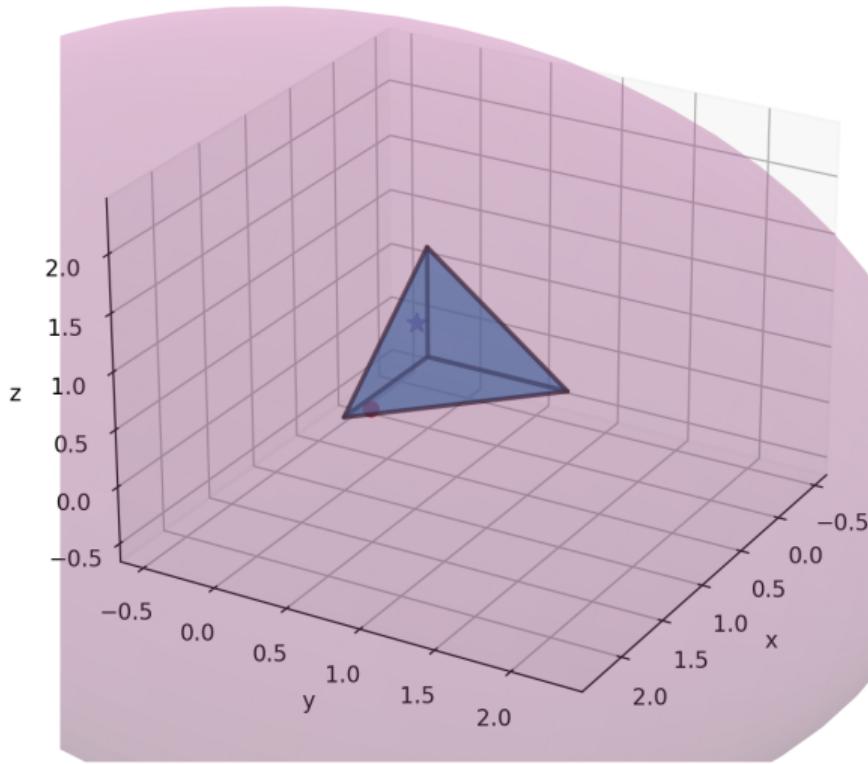
Ellipsoid Method (3D) — iter 4 (show ellipsoid)
feasible center | objective cut | max-axis \approx 3.544 | best $d^T x \approx 0.965$



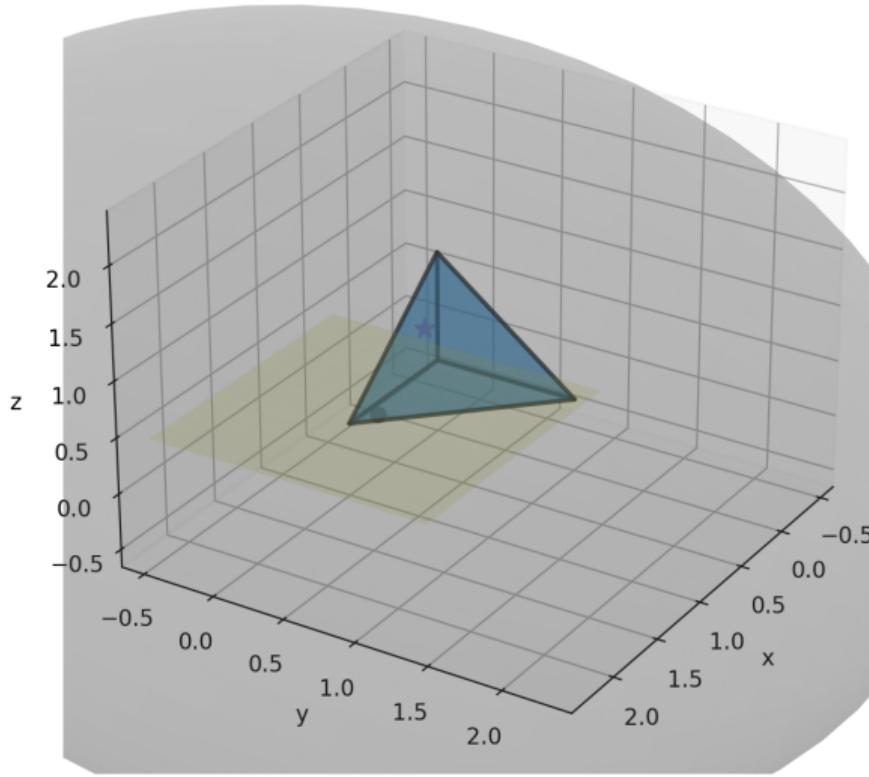
Ellipsoid Method (3D) — iter 4 (show cut plane)
feasible center | objective cut | max-axis \approx 3.544 | best $d^T x \approx 0.965$



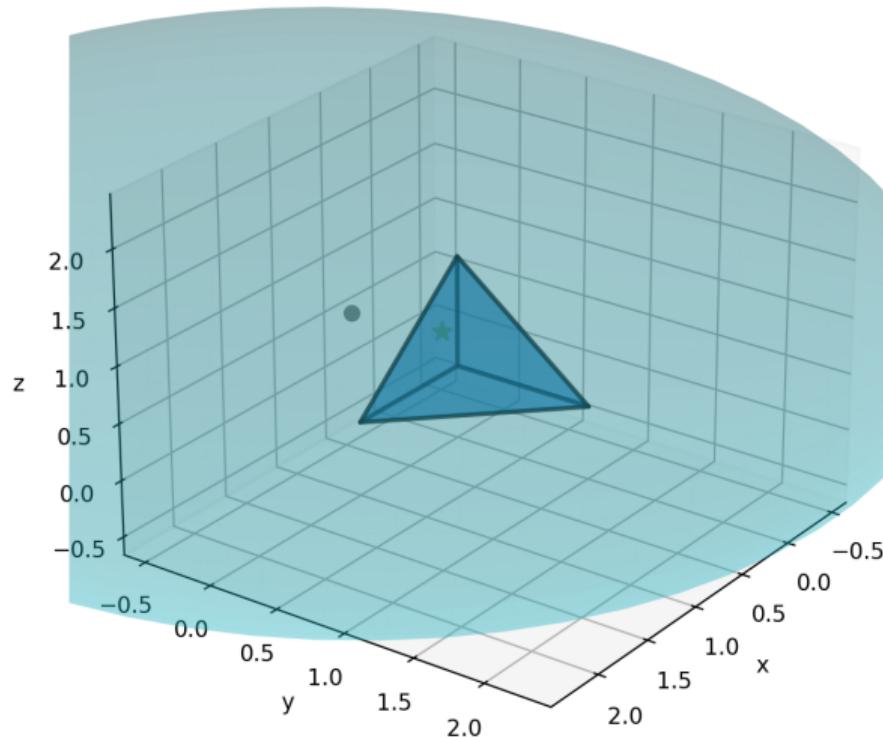
Ellipsoid Method (3D) — iter 5 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 3.759 | best $d^T x \approx 0.965$



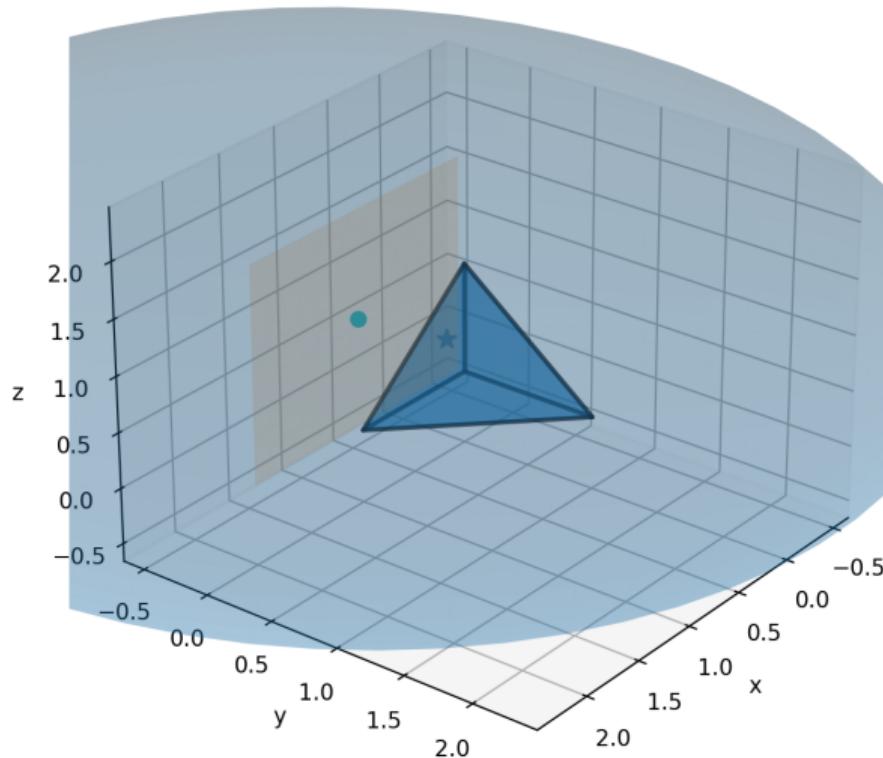
Ellipsoid Method (3D) — iter 5 (show cut plane)
infeasible center | constraint cut | max-axis \approx 3.759 | best $d^T x \approx 0.965$



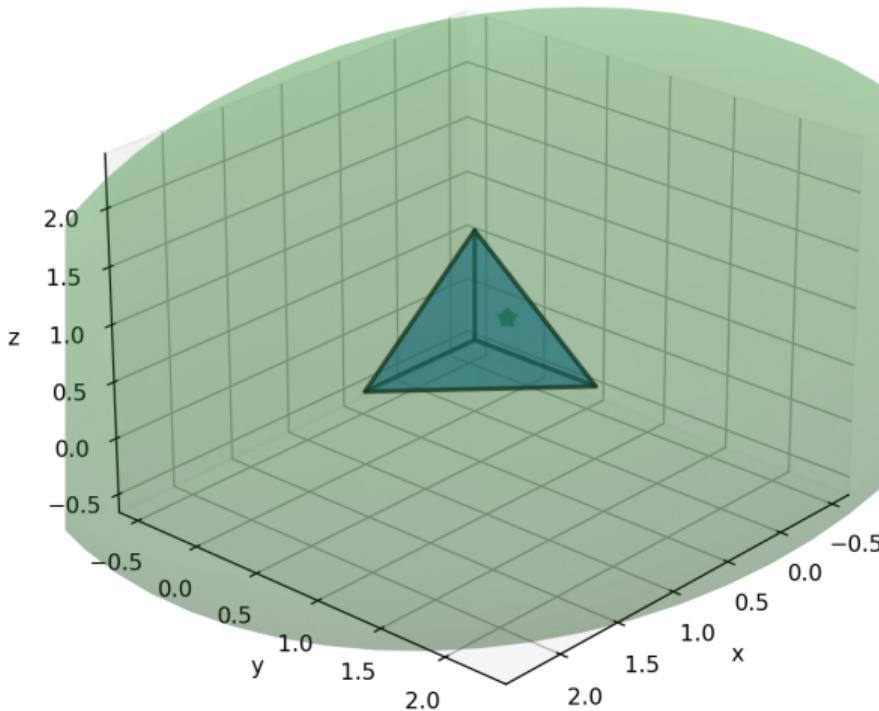
Ellipsoid Method (3D) — iter 6 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 3.805 | best $d^T x \approx 0.965$



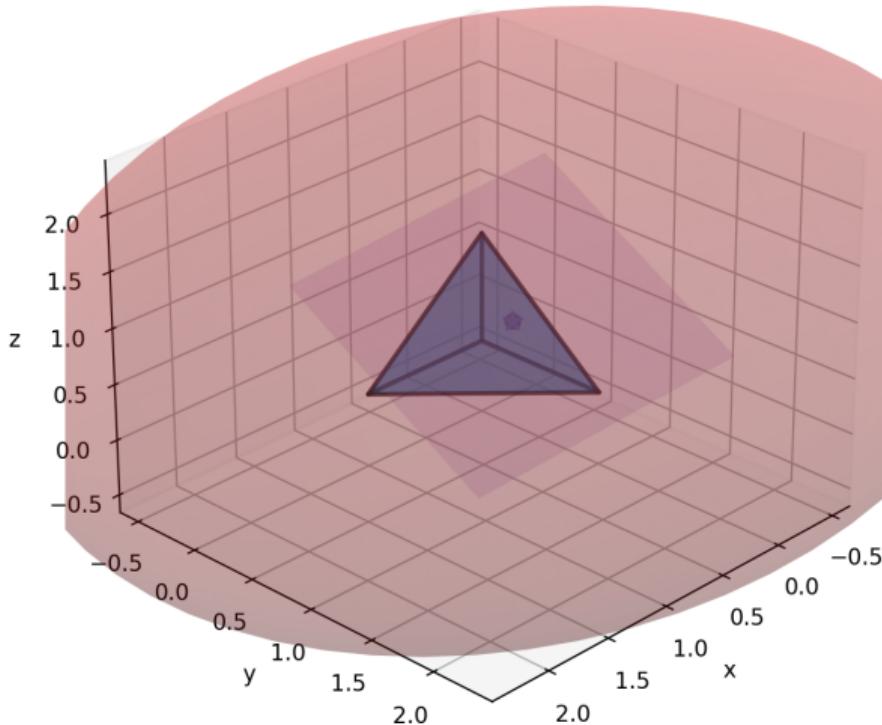
Ellipsoid Method (3D) — iter 6 (show cut plane)
infeasible center | constraint cut | max-axis \approx 3.805 | best $d^T x \approx 0.965$



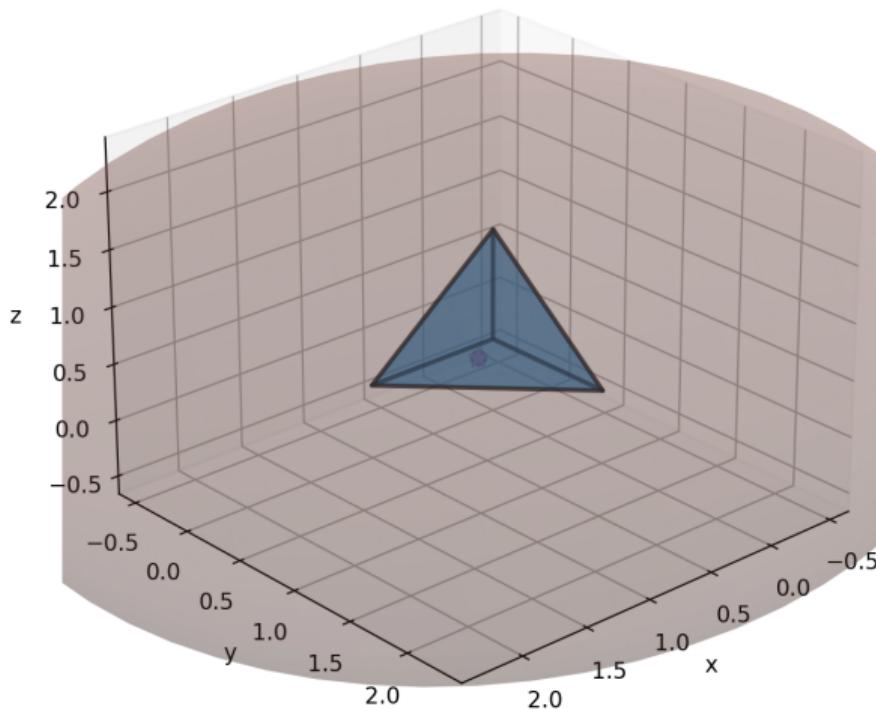
Ellipsoid Method (3D) — iter 7 (show ellipsoid)
feasible center | objective cut | max-axis≈3.183 | best $d^T x \approx 0.800$



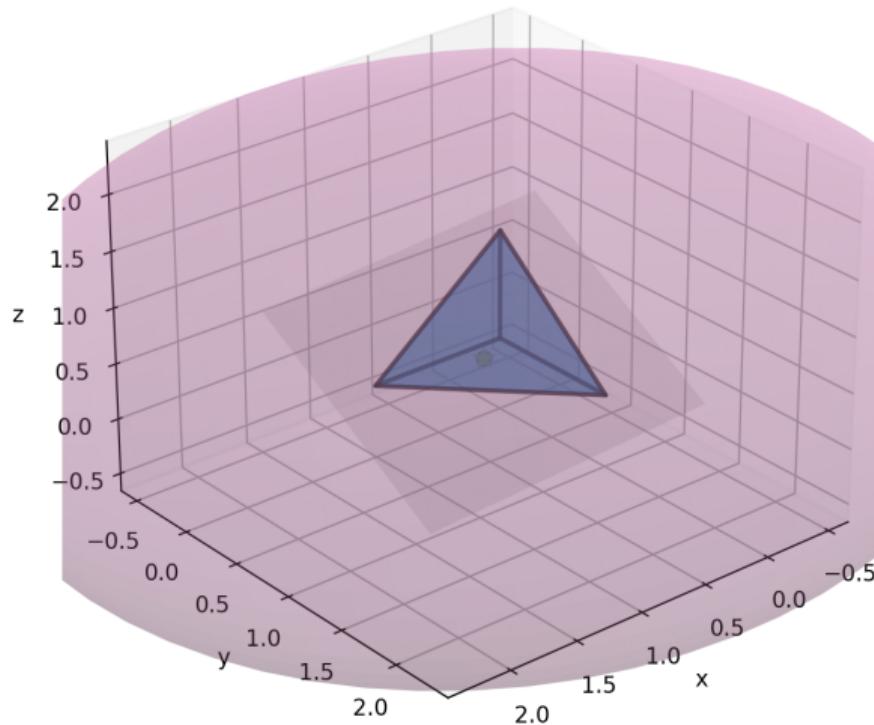
Ellipsoid Method (3D) — iter 7 (show cut plane)
feasible center | objective cut | max-axis \approx 3.183 | best $d^T x \approx 0.800$



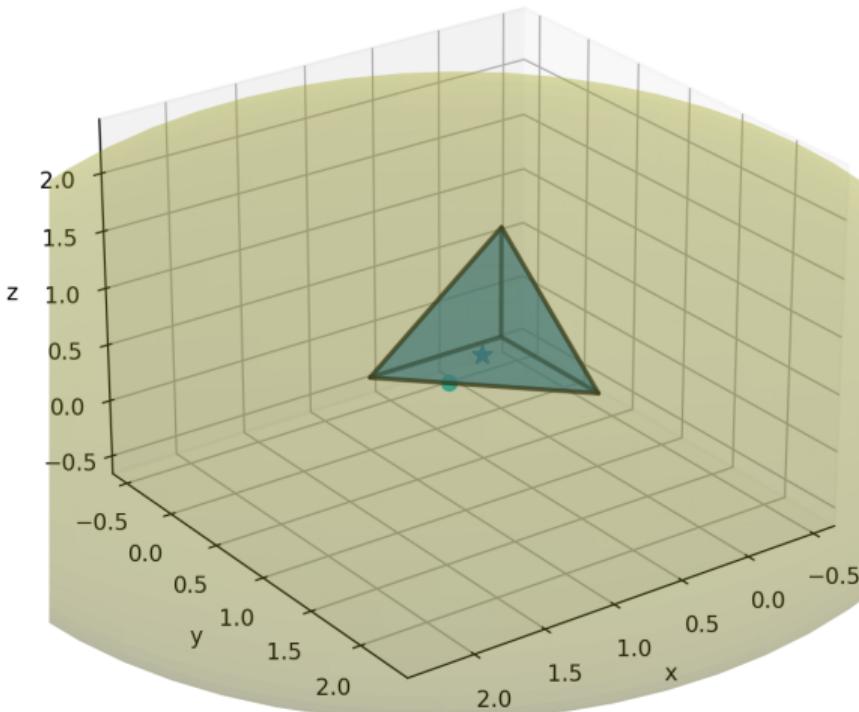
Ellipsoid Method (3D) — iter 8 (show ellipsoid)
feasible center | objective cut | max-axis≈3.240 | best $d^T x \approx 0.344$



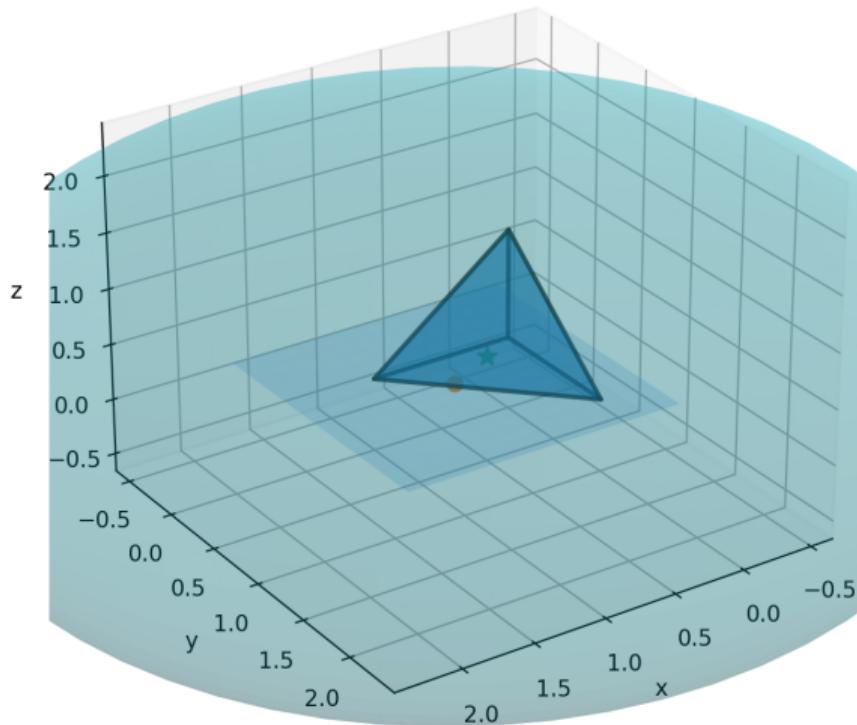
Ellipsoid Method (3D) — iter 8 (show cut plane)
feasible center | objective cut | max-axis \approx 3.240 | best $d^T x \approx 0.344$



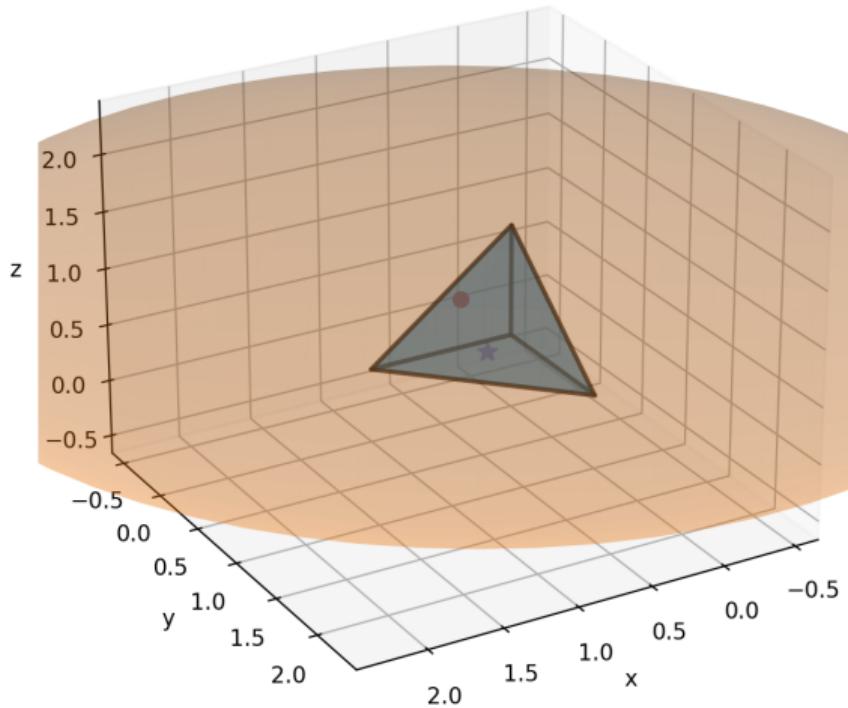
Ellipsoid Method (3D) — iter 9 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 3.375 | best $d^T x \approx 0.344$



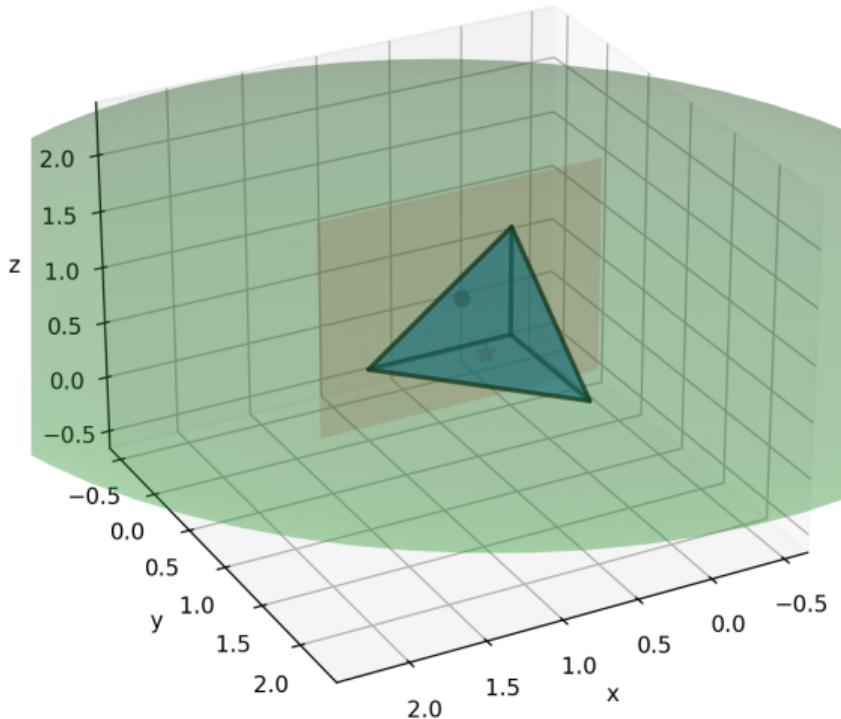
Ellipsoid Method (3D) — iter 9 (show cut plane)
infeasible center | constraint cut | max-axis \approx 3.375 | best $d^T x \approx 0.344$



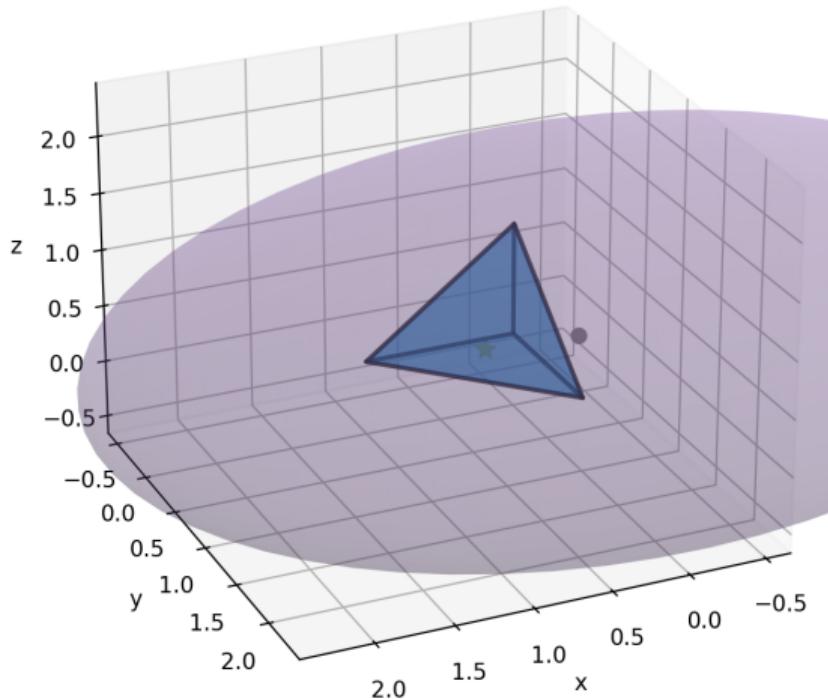
Ellipsoid Method (3D) — iter 10 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 3.579 | best $d^T x \approx 0.344$



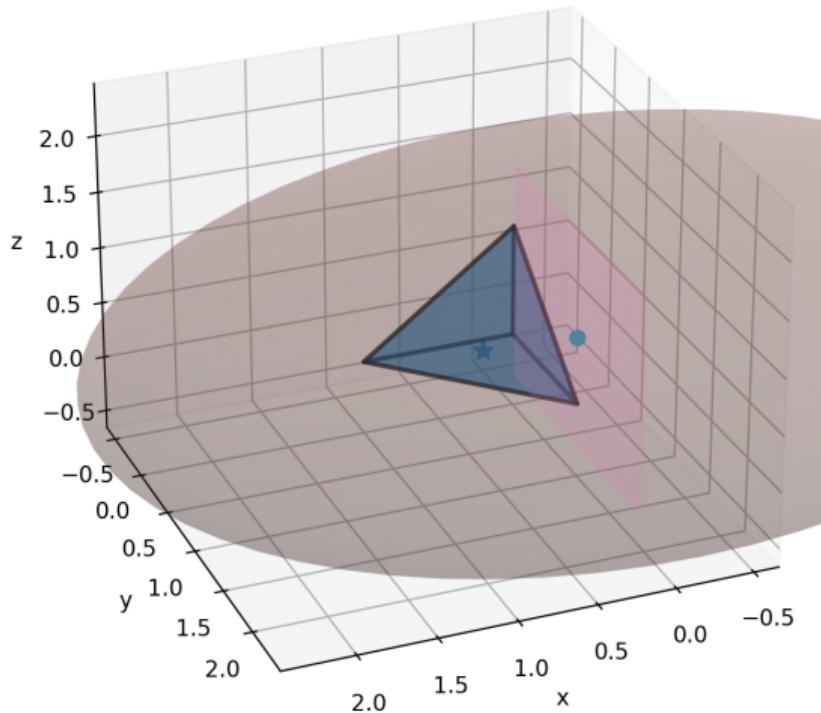
Ellipsoid Method (3D) — iter 10 (show cut plane)
infeasible center | constraint cut | max-axis \approx 3.579 | best $d^T x \approx 0.344$



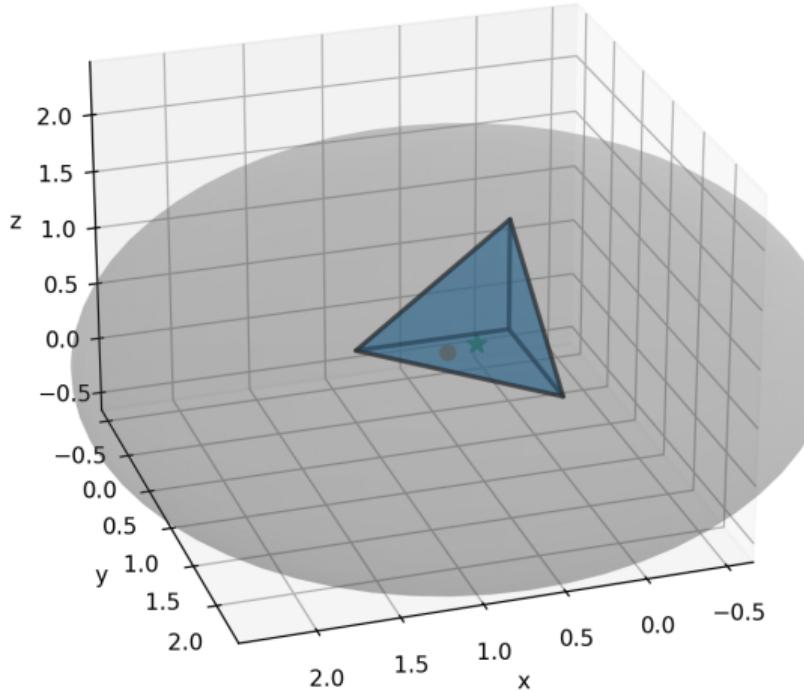
Ellipsoid Method (3D) — iter 11 (show ellipsoid)
infeasible center | constraint cut | max-axis≈3.162 | best $d^T x \approx 0.344$



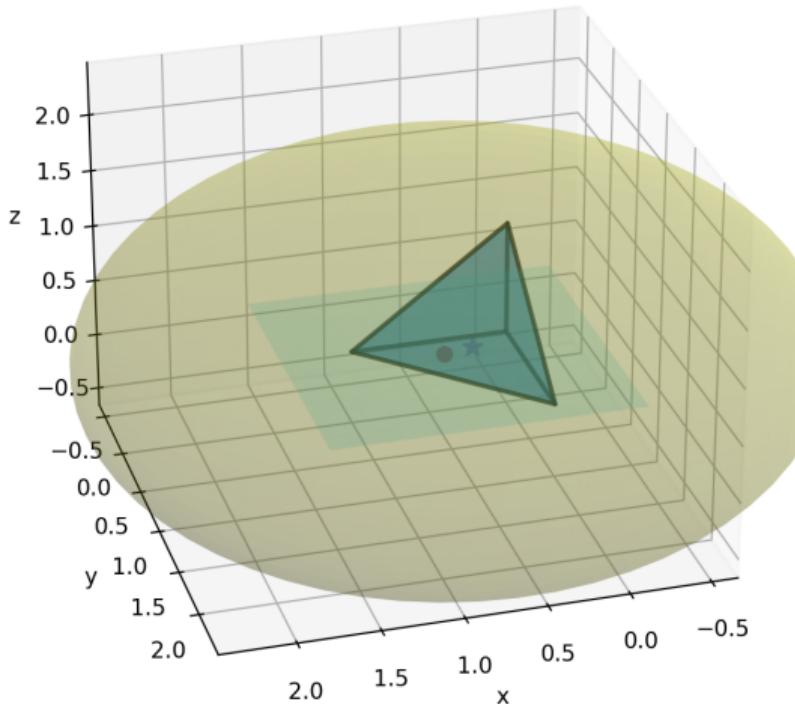
Ellipsoid Method (3D) — iter 11 (show cut plane)
infeasible center | constraint cut | max-axis≈3.162 | best $d^T x \approx 0.344$



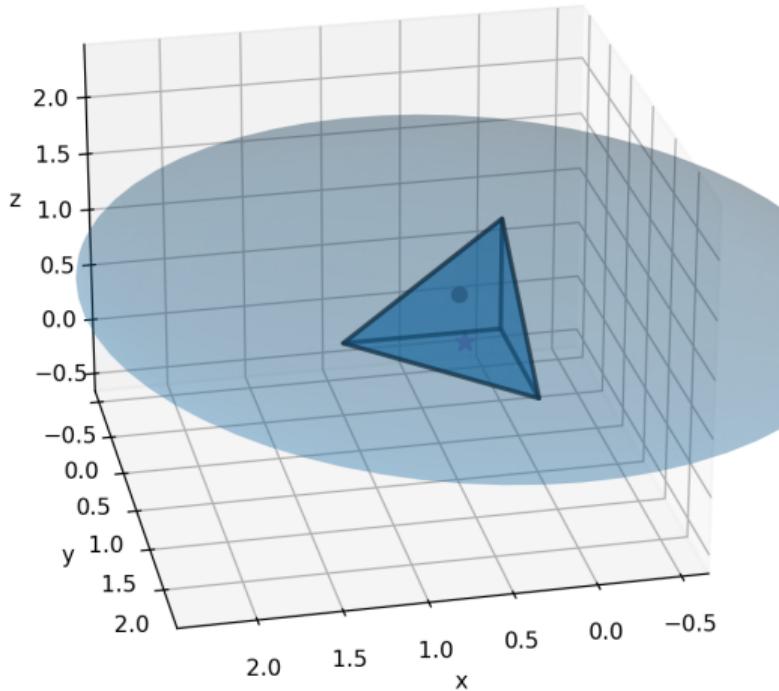
Ellipsoid Method (3D) — iter 12 (show ellipsoid)
infeasible center | constraint cut | max-axis≈2.399 | best $d^T x \approx 0.344$



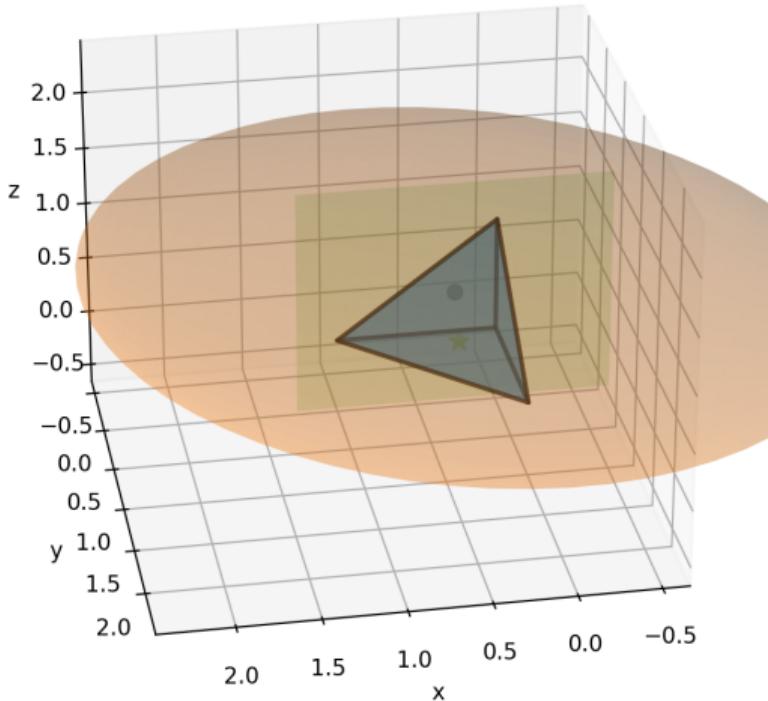
Ellipsoid Method (3D) — iter 12 (show cut plane)
infeasible center | constraint cut | max-axis≈2.399 | best $d^T x \approx 0.344$



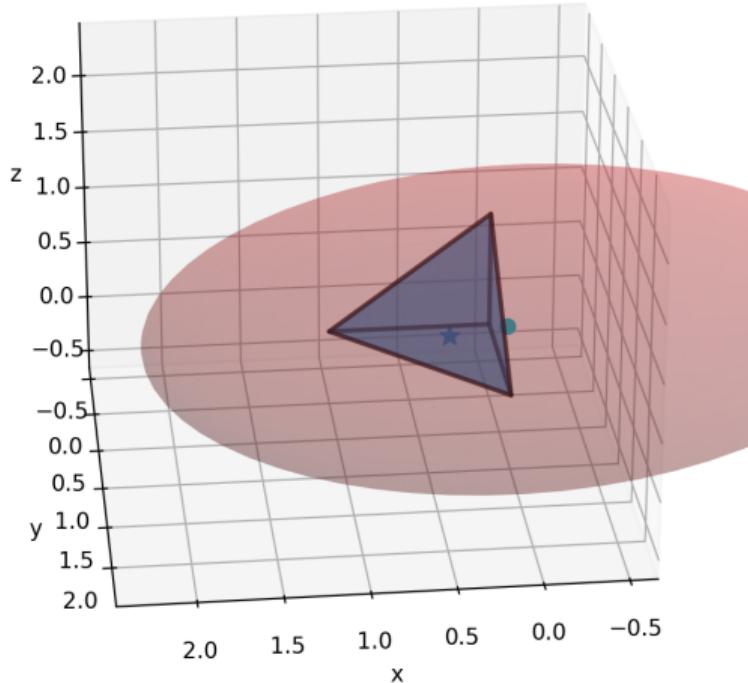
Ellipsoid Method (3D) — iter 13 (show ellipsoid)
infeasible center | constraint cut | max-axis≈2.535 | best $d^T x \approx 0.344$



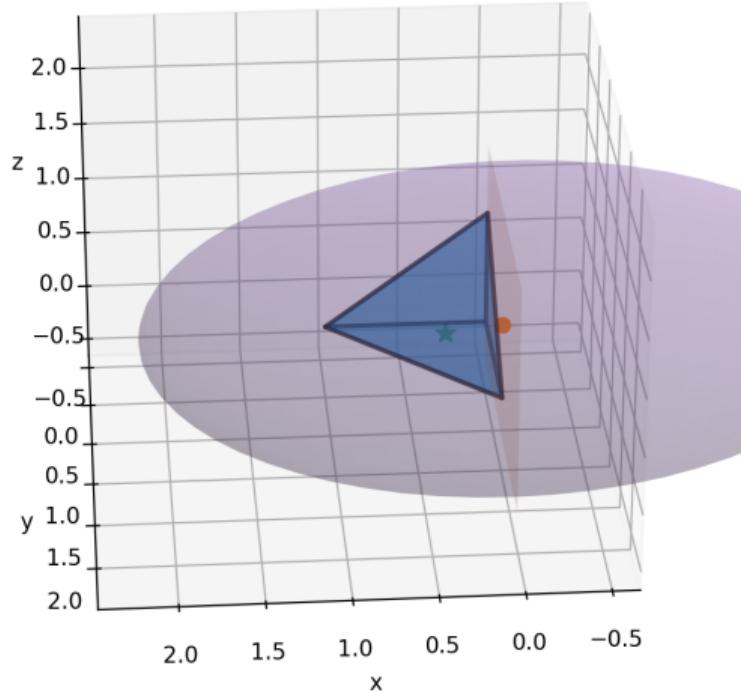
Ellipsoid Method (3D) — iter 13 (show cut plane)
infeasible center | constraint cut | max-axis≈2.535 | best $d^T x \approx 0.344$



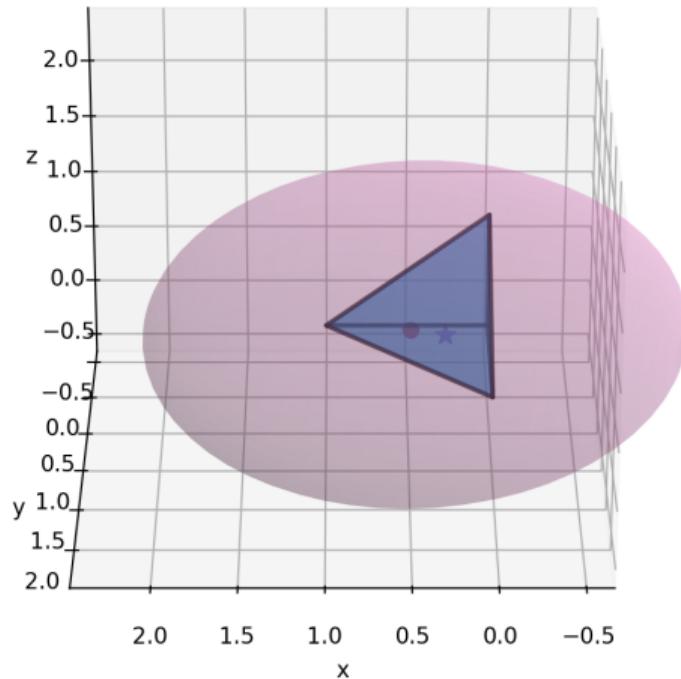
Ellipsoid Method (3D) — iter 14 (show ellipsoid)
infeasible center | constraint cut | max-axis≈2.341 | best $d^T x \approx 0.344$



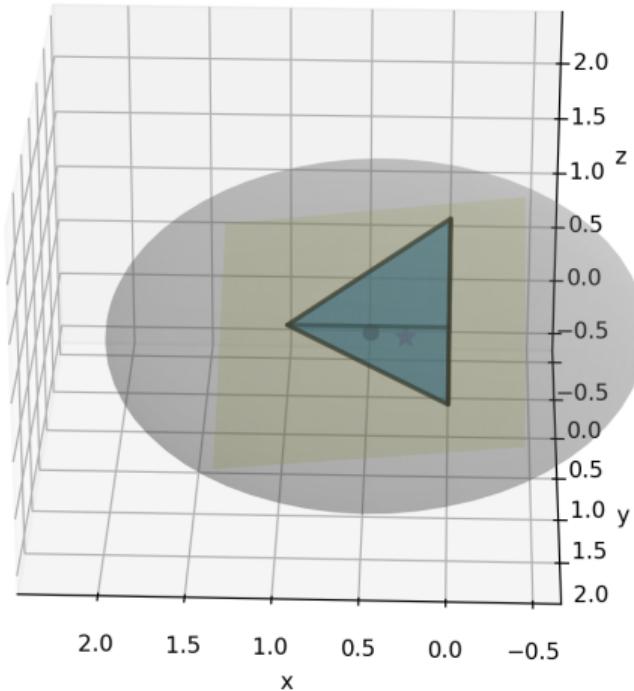
Ellipsoid Method (3D) — iter 14 (show cut plane)
infeasible center | constraint cut | max-axis≈2.341 | best $d^T x \approx 0.344$



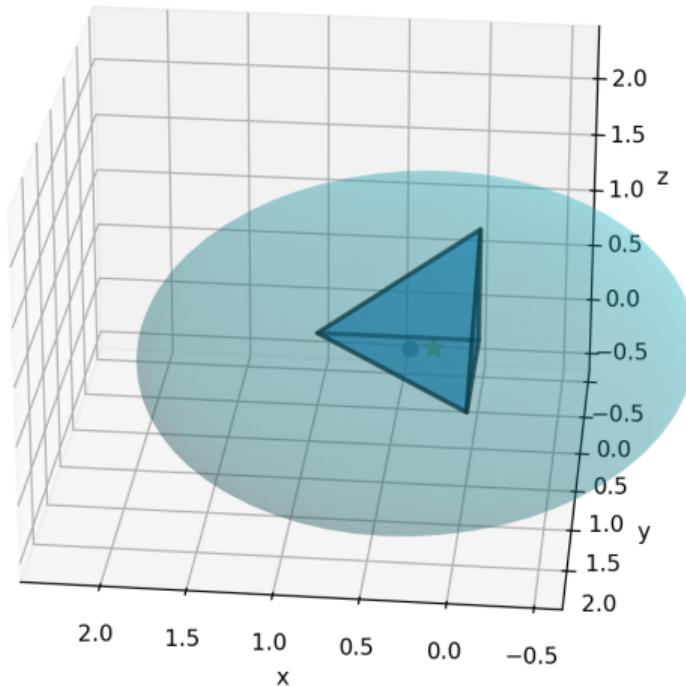
Ellipsoid Method (3D) — iter 15 (show ellipsoid)
feasible center | objective cut | max-axis \approx 1.772 | best $d^T x \approx 0.344$



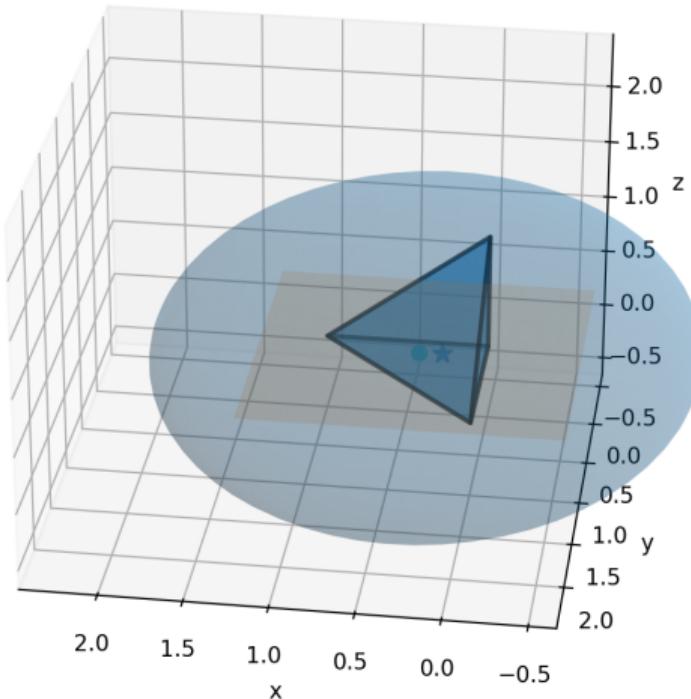
Ellipsoid Method (3D) — iter 15 (show cut plane)
feasible center | objective cut | max-axis \approx 1.772 | best $d^T x \approx 0.344$



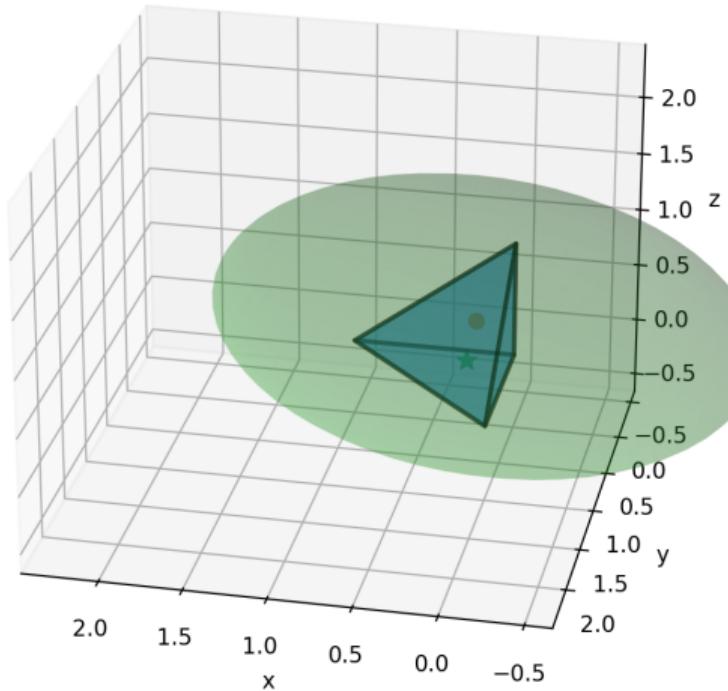
Ellipsoid Method (3D) — iter 16 (show ellipsoid)
infeasible center | constraint cut | max-axis≈1.879 | best $d^T x \approx 0.344$



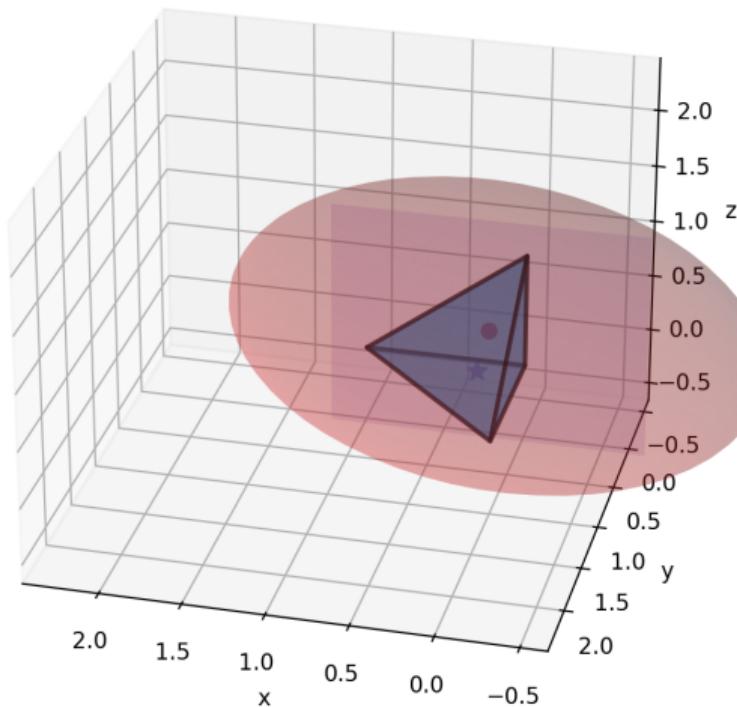
Ellipsoid Method (3D) — iter 16 (show cut plane)
infeasible center | constraint cut | max-axis \approx 1.879 | best $d^T x \approx 0.344$



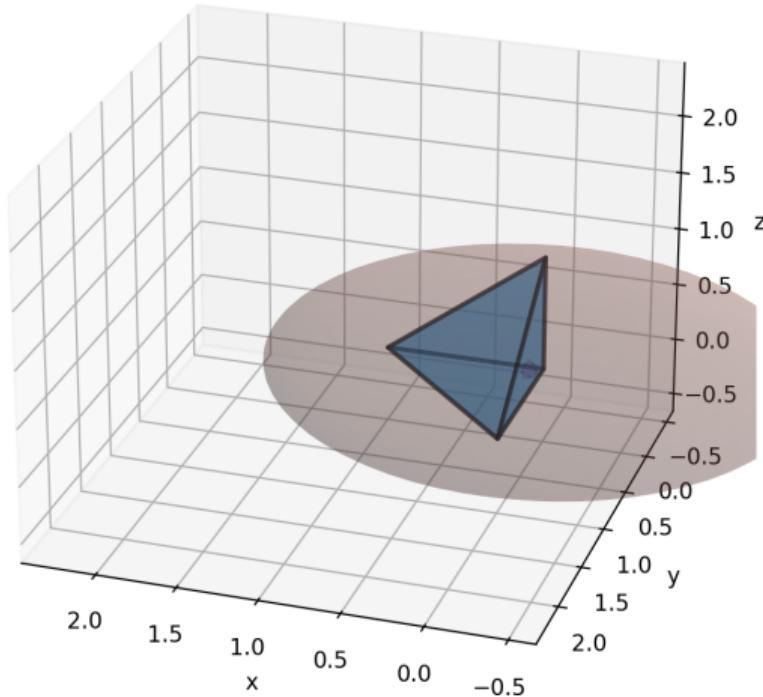
Ellipsoid Method (3D) — iter 17 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 1.978 | best $d^T x \approx 0.344$



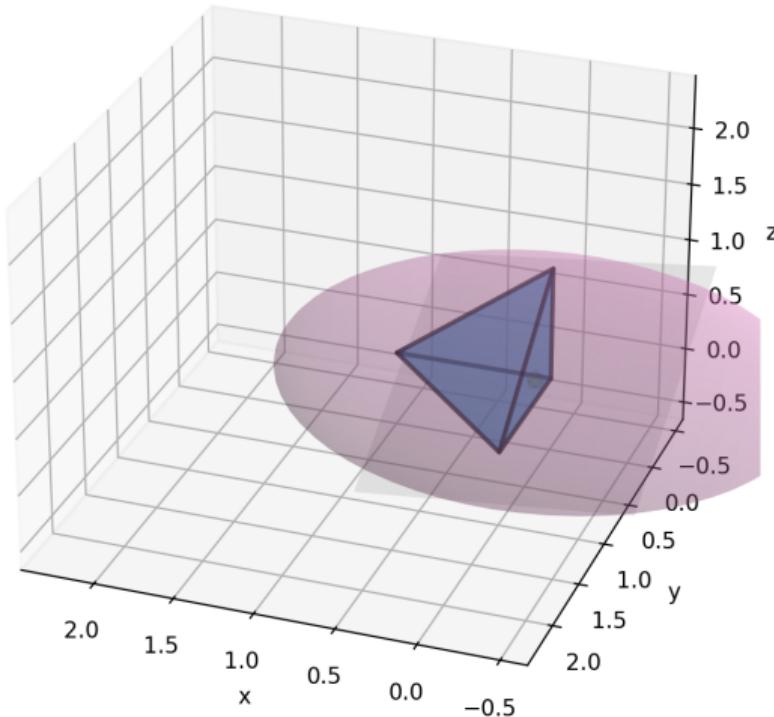
Ellipsoid Method (3D) — iter 17 (show cut plane)
infeasible center | constraint cut | max-axis \approx 1.978 | best $d^T x \approx 0.344$



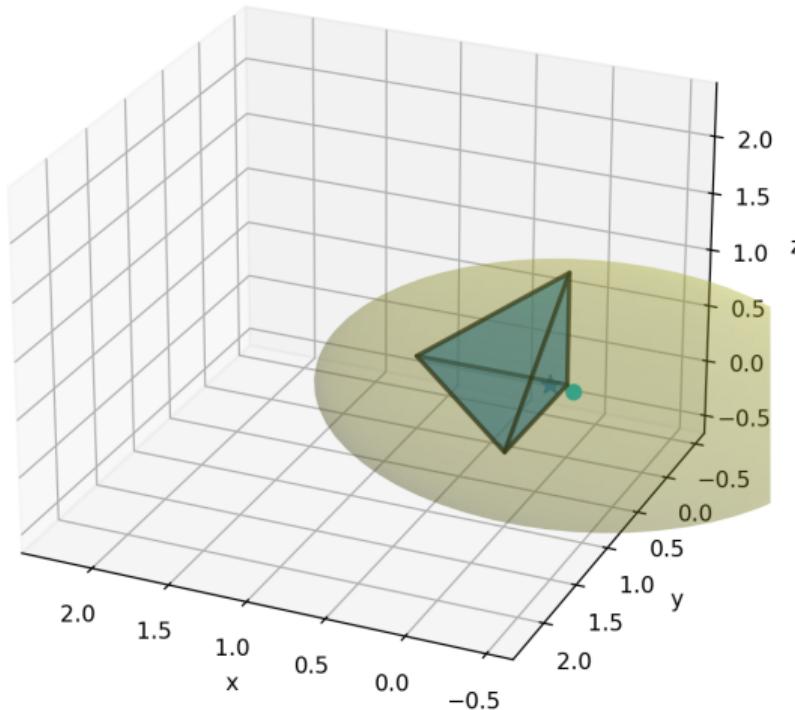
Ellipsoid Method (3D) — iter 18 (show ellipsoid)
feasible center | objective cut | max-axis \approx 1.876 | best $d^T x \approx 0.318$



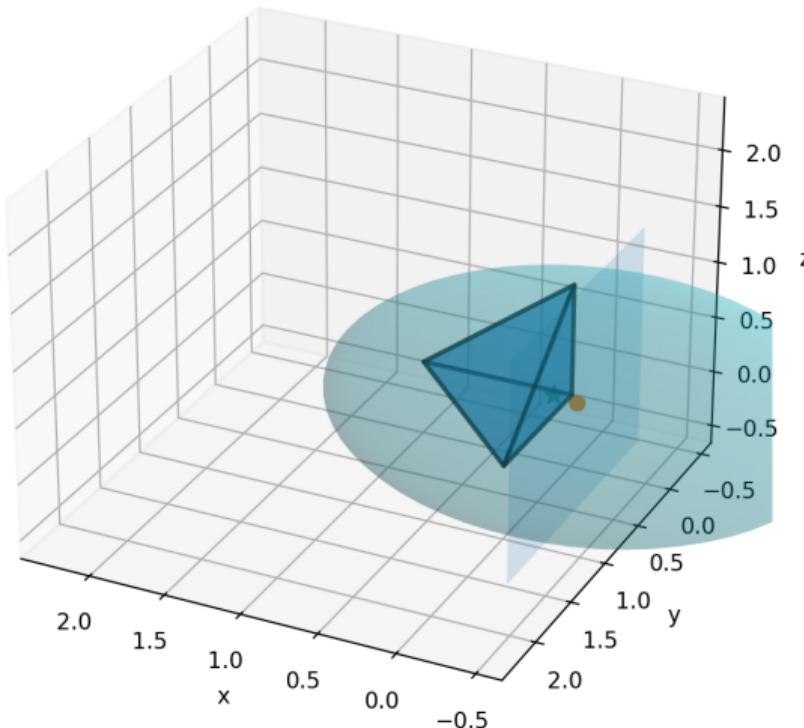
Ellipsoid Method (3D) — iter 18 (show cut plane)
feasible center | objective cut | max-axis \approx 1.876 | best $d^T x \approx 0.318$



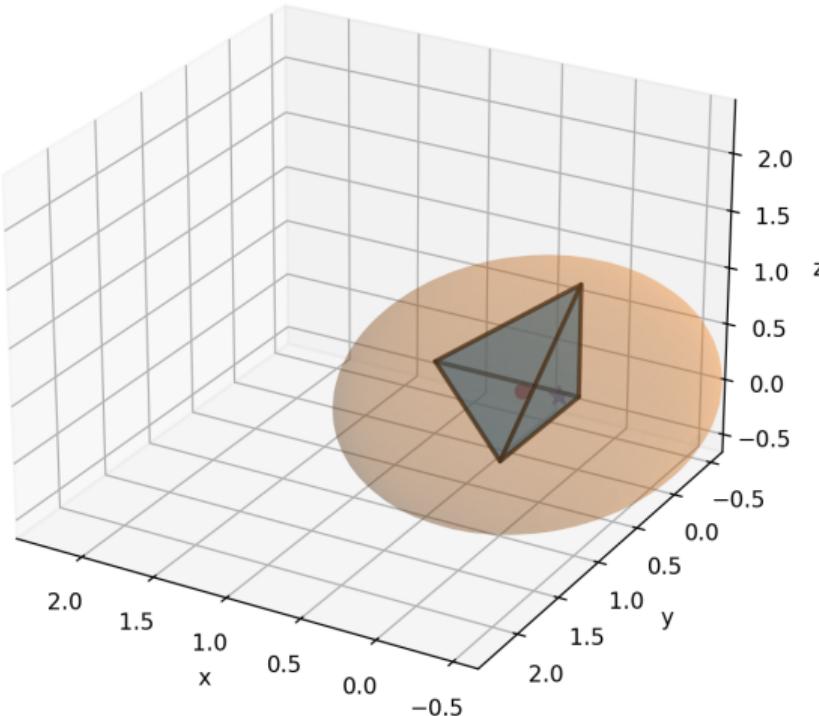
Ellipsoid Method (3D) — iter 19 (show ellipsoid)
infeasible center | constraint cut | max-axis≈1.971 | best $d^T x \approx 0.318$



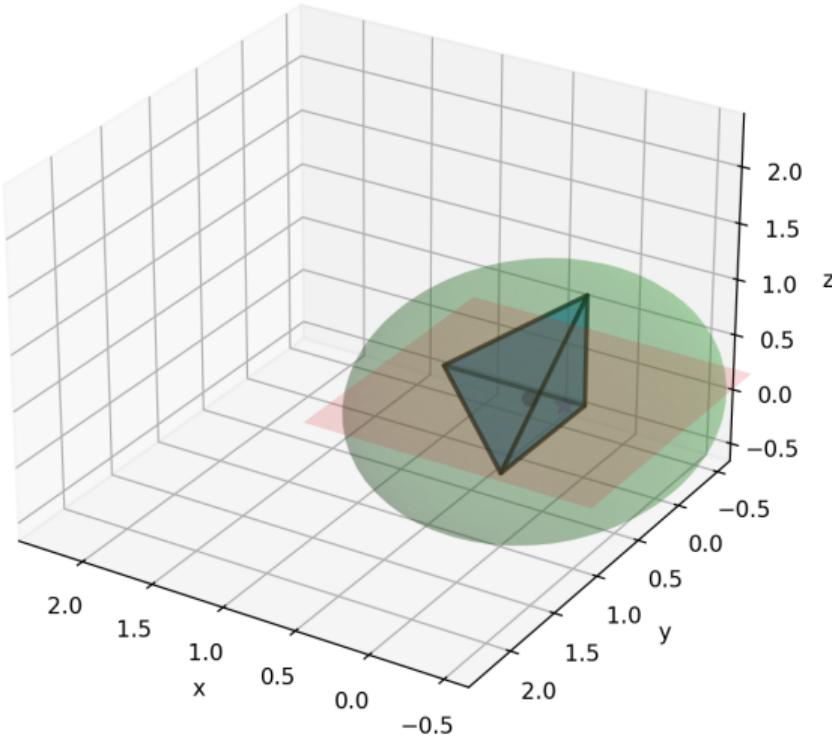
Ellipsoid Method (3D) — iter 19 (show cut plane)
infeasible center | constraint cut | max-axis \approx 1.971 | best $d^T x \approx 0.318$



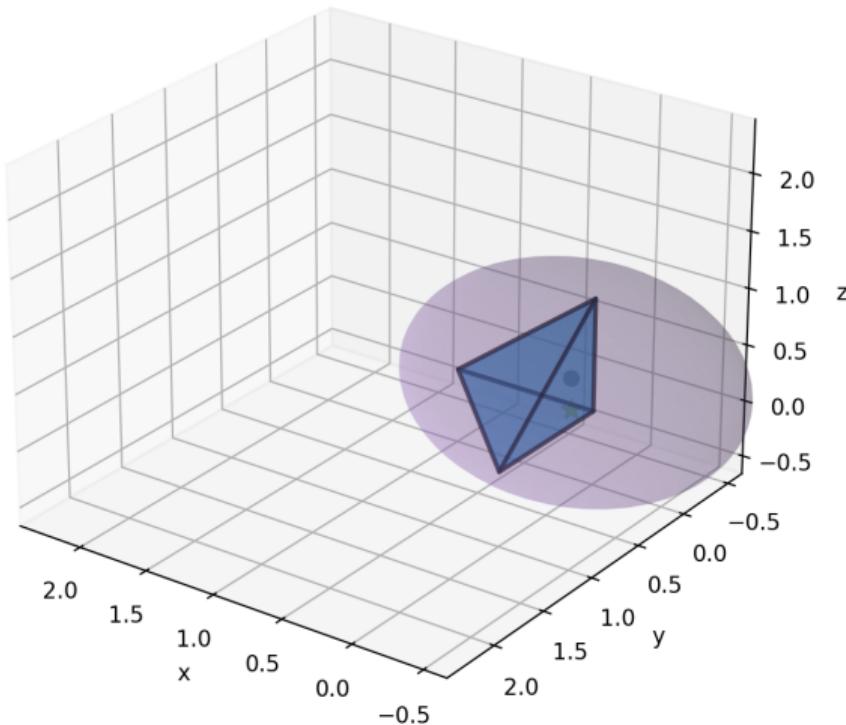
Ellipsoid Method (3D) — iter 20 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 1.486 | best $d^T x \approx 0.318$



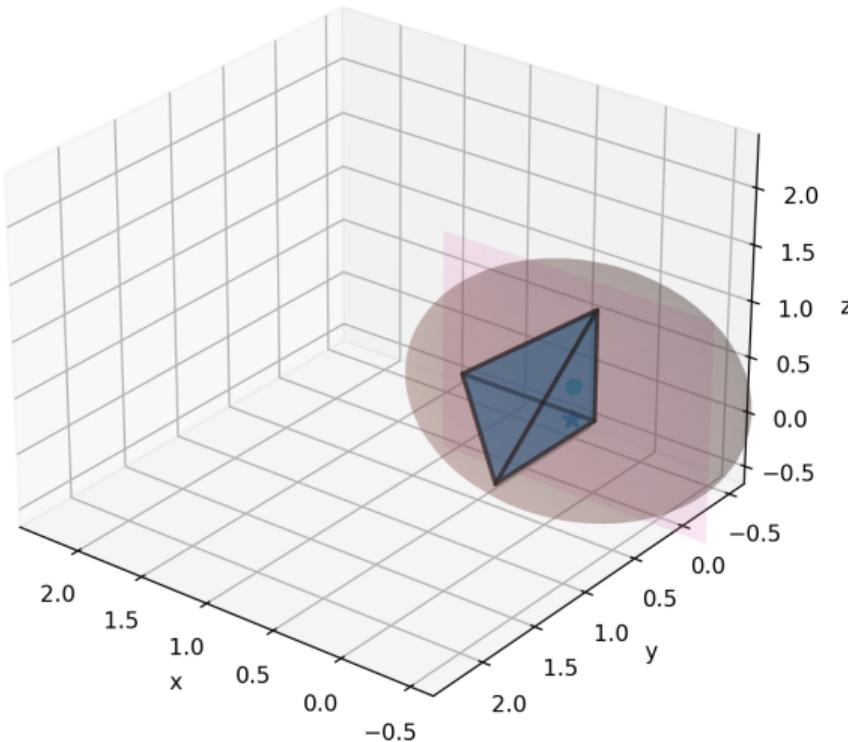
Ellipsoid Method (3D) — iter 20 (show cut plane)
infeasible center | constraint cut | max-axis \approx 1.486 | best $d^T x \approx 0.318$



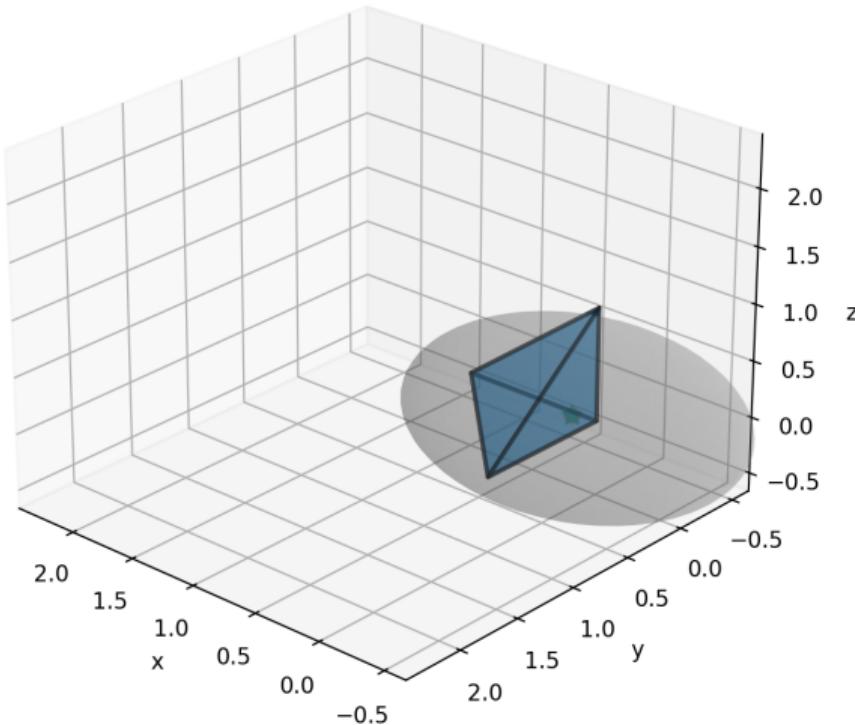
Ellipsoid Method (3D) — iter 21 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 1.557 | best $d^T x \approx 0.318$



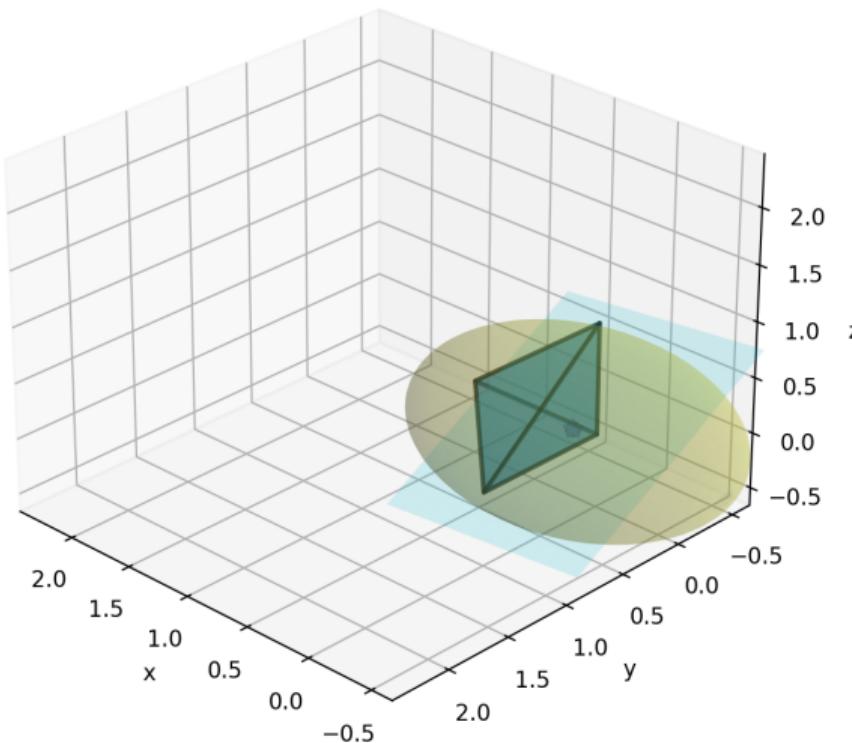
Ellipsoid Method (3D) — iter 21 (show cut plane)
infeasible center | constraint cut | max-axis \approx 1.557 | best $d^T x \approx 0.318$



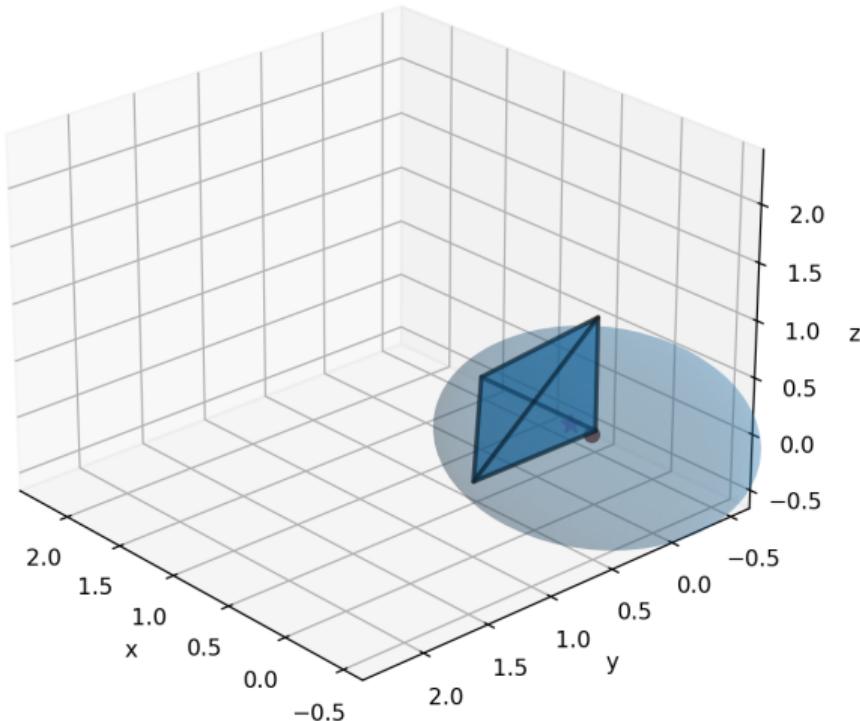
Ellipsoid Method (3D) — iter 22 (show ellipsoid)
feasible center | objective cut | max-axis \approx 1.499 | best $d^T x \approx 0.252$



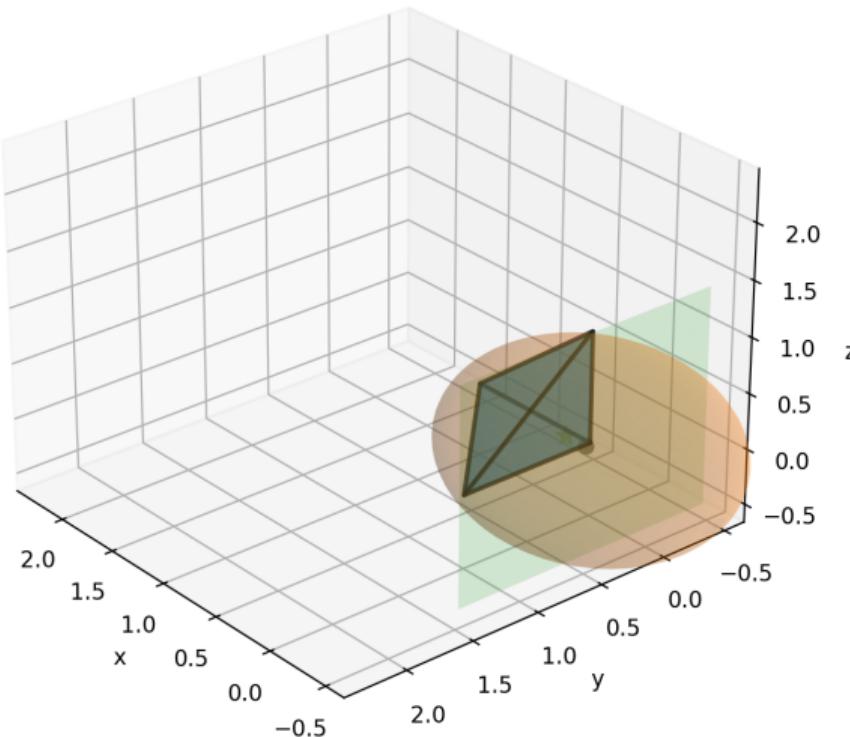
Ellipsoid Method (3D) — iter 22 (show cut plane)
feasible center | objective cut | max-axis \approx 1.499 | best $d^T x \approx 0.252$



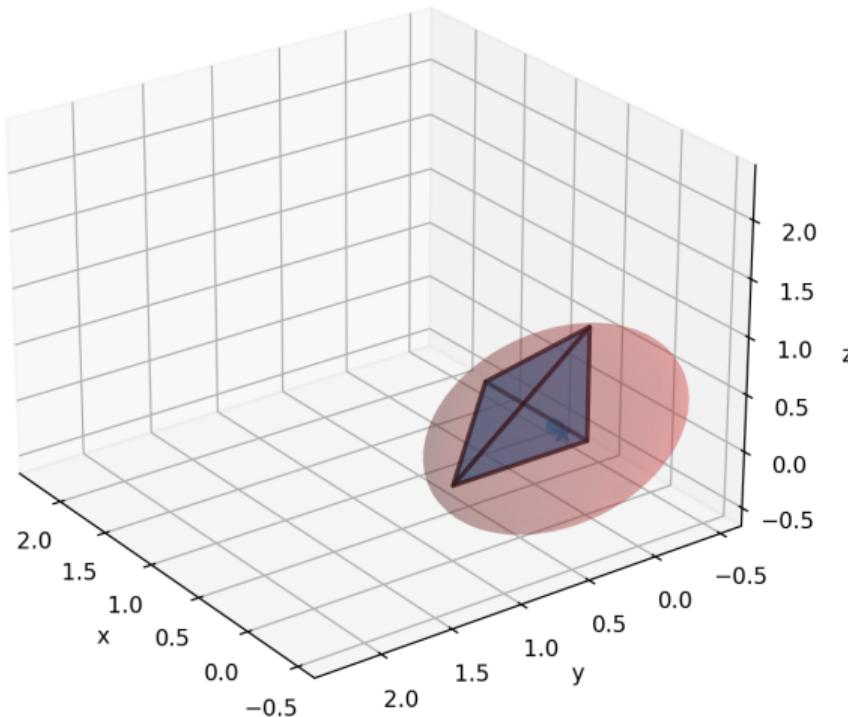
Ellipsoid Method (3D) — iter 23 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 1.564 | best $d^T x \approx 0.252$



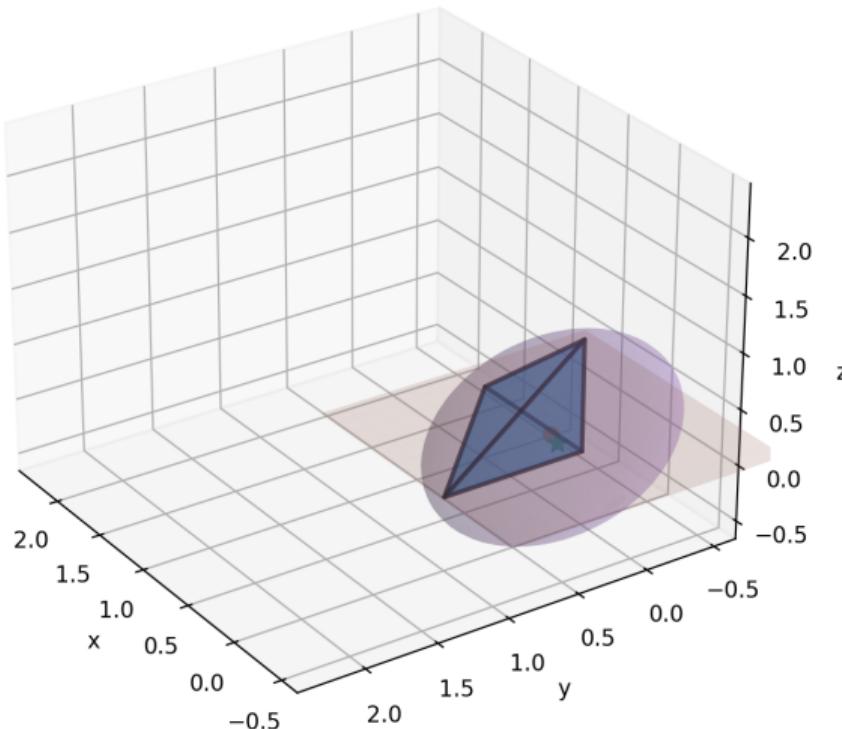
Ellipsoid Method (3D) — iter 23 (show cut plane)
infeasible center | constraint cut | max-axis \approx 1.564 | best $d^T x \approx 0.252$



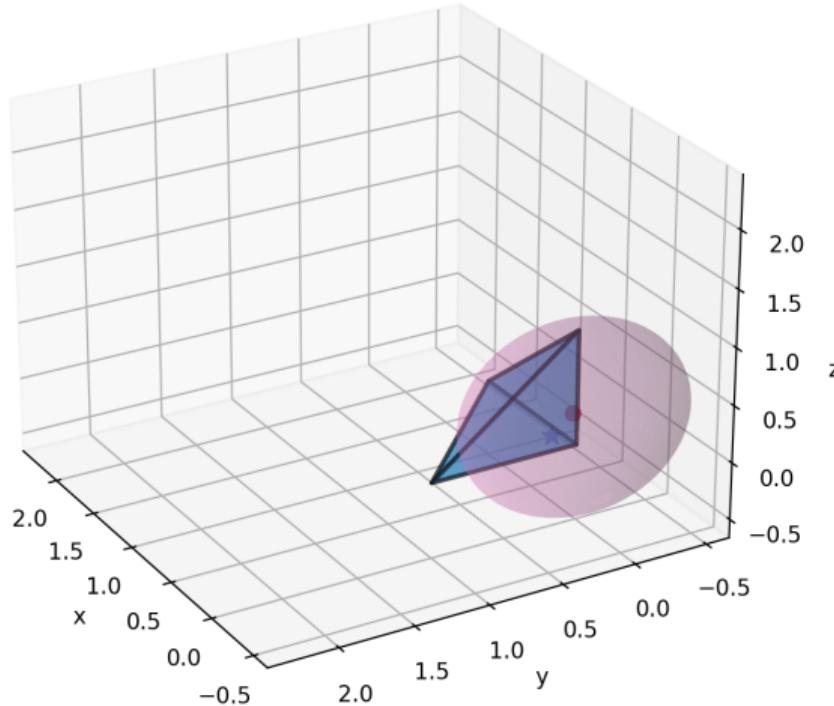
Ellipsoid Method (3D) — iter 24 (show ellipsoid)
infeasible center | constraint cut | max-axis≈1.178 | best $d^T x \approx 0.252$



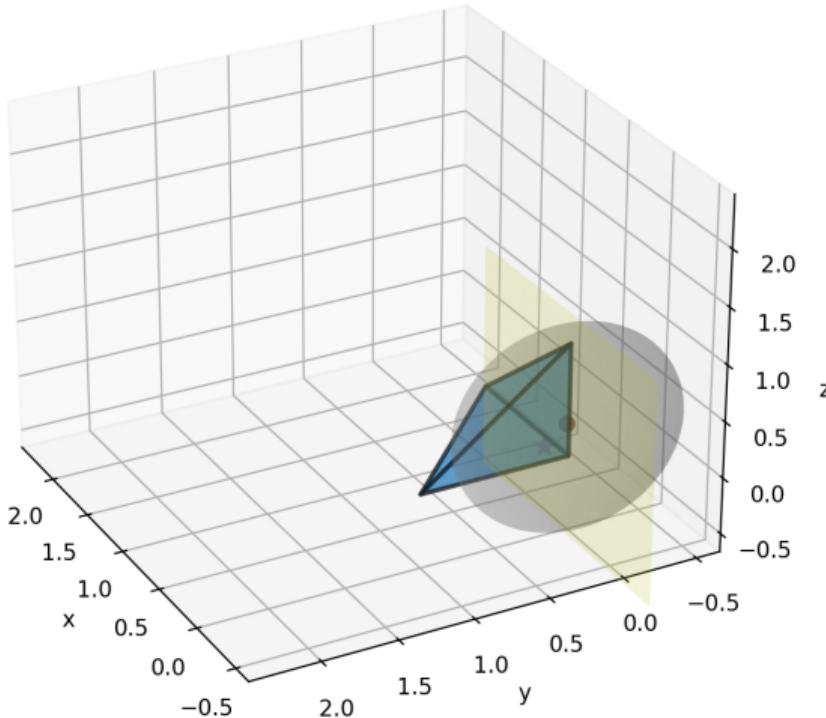
Ellipsoid Method (3D) — iter 24 (show cut plane)
infeasible center | constraint cut | max-axis \approx 1.178 | best $d^T x \approx 0.252$



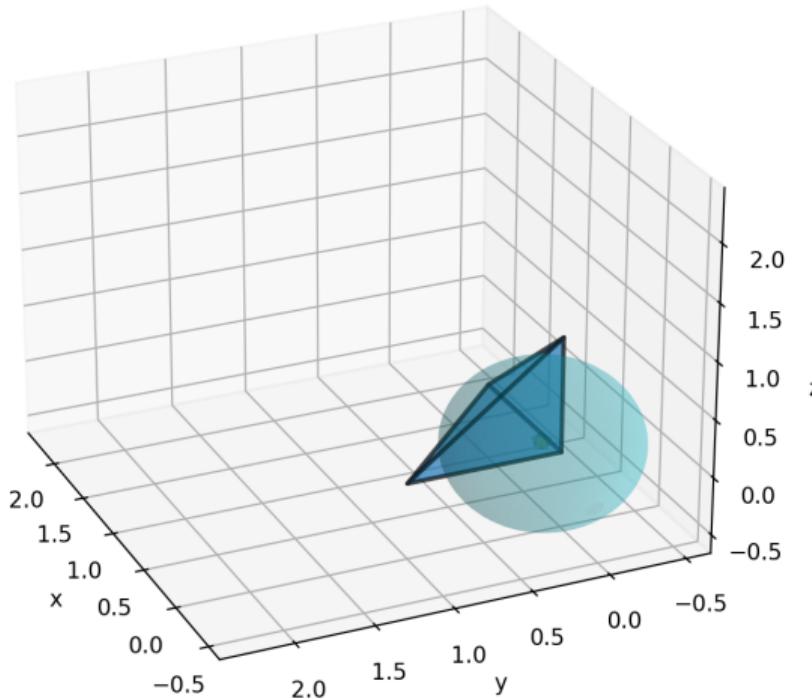
Ellipsoid Method (3D) — iter 25 (show ellipsoid)
infeasible center | constraint cut | max-axis≈1.232 | best $d^T x \approx 0.252$



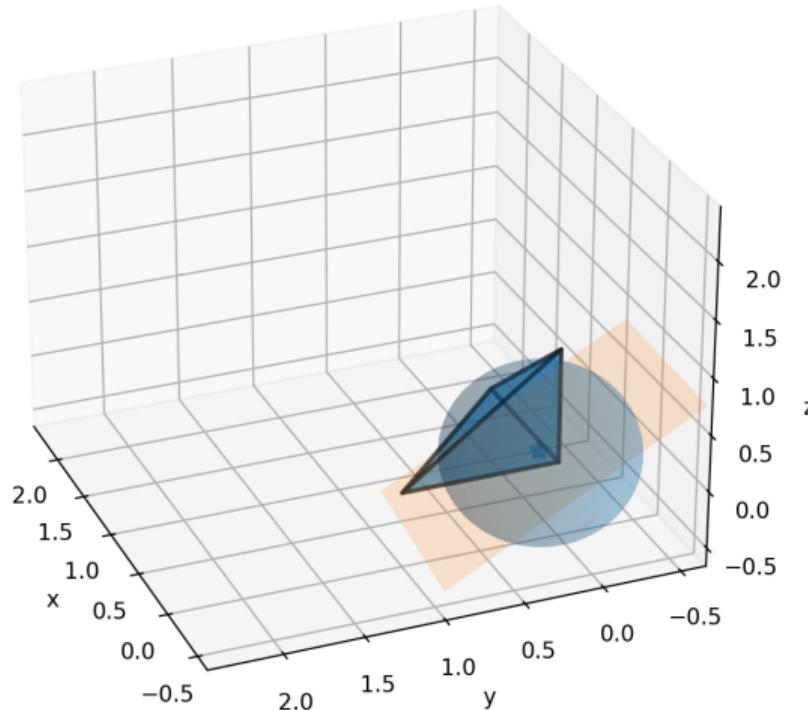
Ellipsoid Method (3D) — iter 25 (show cut plane)
infeasible center | constraint cut | max-axis≈1.232 | best $d^T x \approx 0.252$



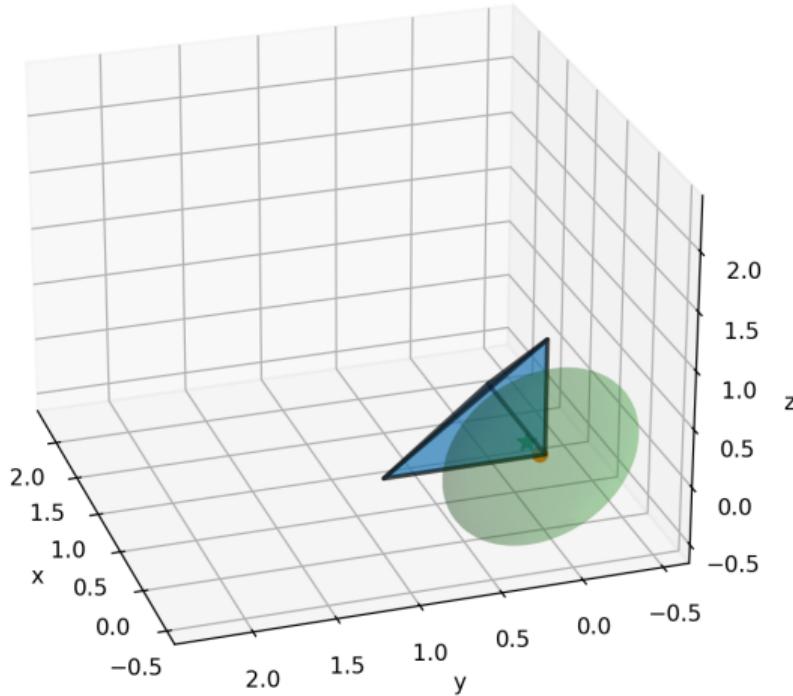
Ellipsoid Method (3D) — iter 26 (show ellipsoid)
feasible center | objective cut | max-axis \approx 1.196 | best $d^T x \approx 0.205$



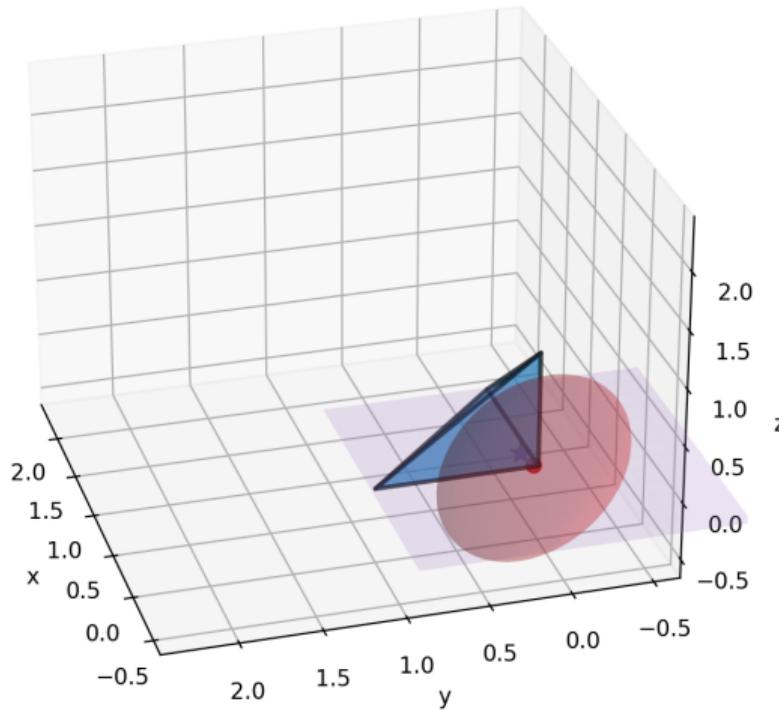
Ellipsoid Method (3D) — iter 26 (show cut plane)
feasible center | objective cut | max-axis \approx 1.196 | best $d^T x \approx 0.205$



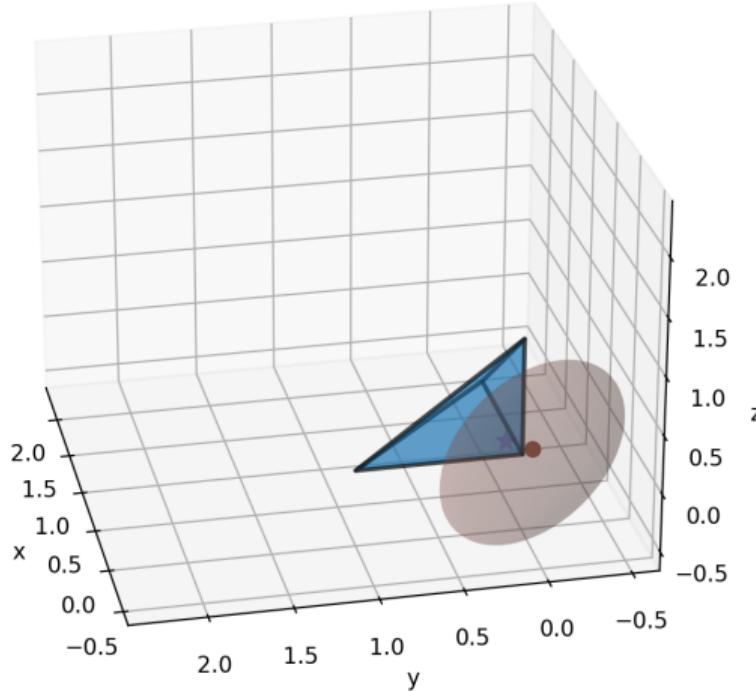
Ellipsoid Method (3D) — iter 27 (show ellipsoid)
infeasible center | constraint cut | max-axis≈1.242 | best $d^T x \approx 0.205$



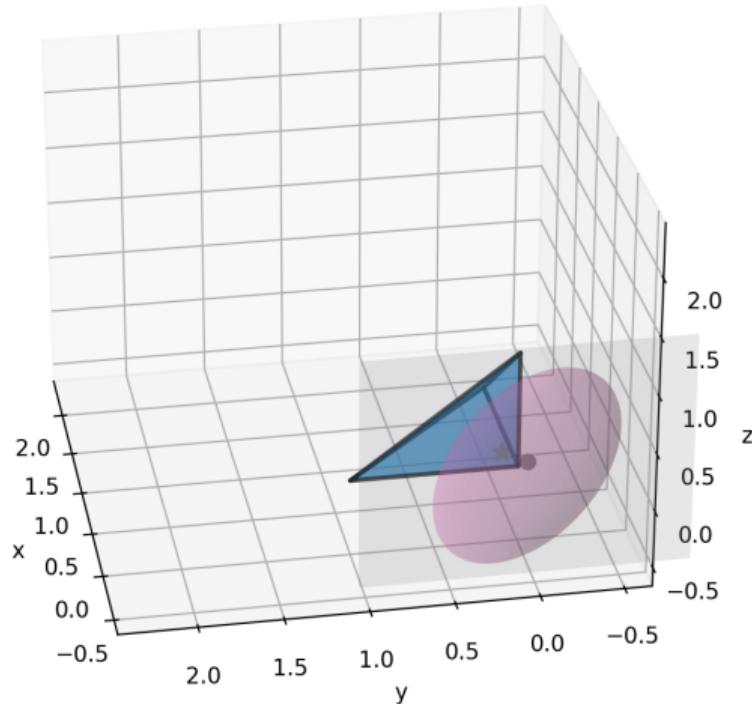
Ellipsoid Method (3D) — iter 27 (show cut plane)
infeasible center | constraint cut | max-axis \approx 1.242 | best $d^T x \approx 0.205$



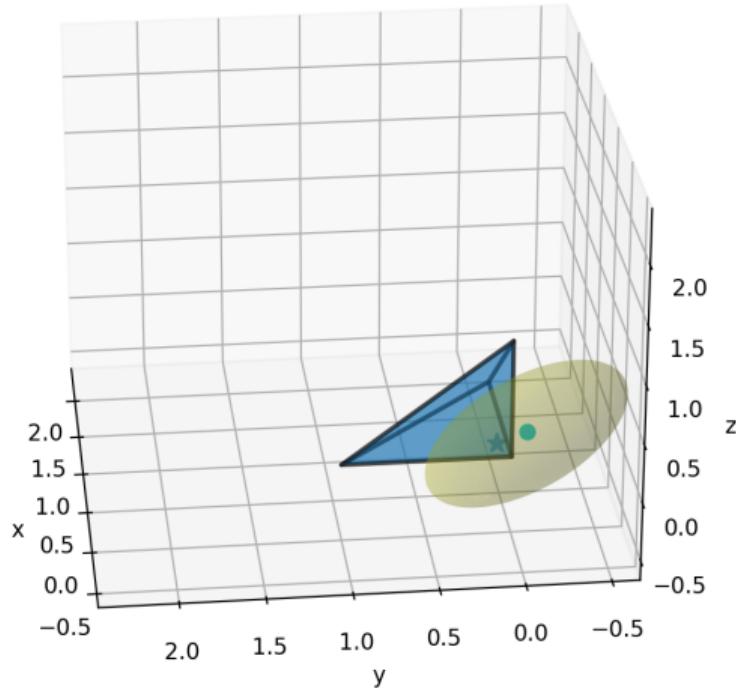
Ellipsoid Method (3D) — iter 28 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 1.253 | best $d^T x \approx 0.205$



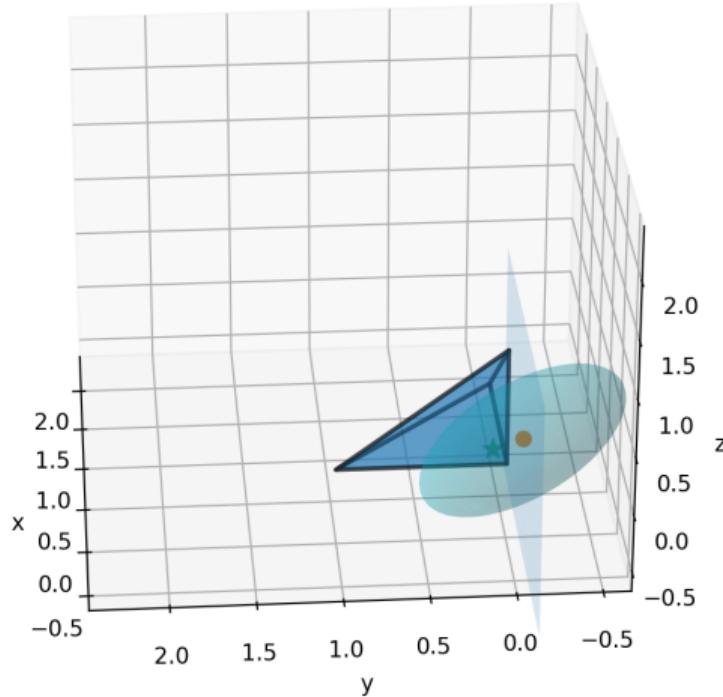
Ellipsoid Method (3D) — iter 28 (show cut plane)
infeasible center | constraint cut | max-axis \approx 1.253 | best $d^T x \approx 0.205$



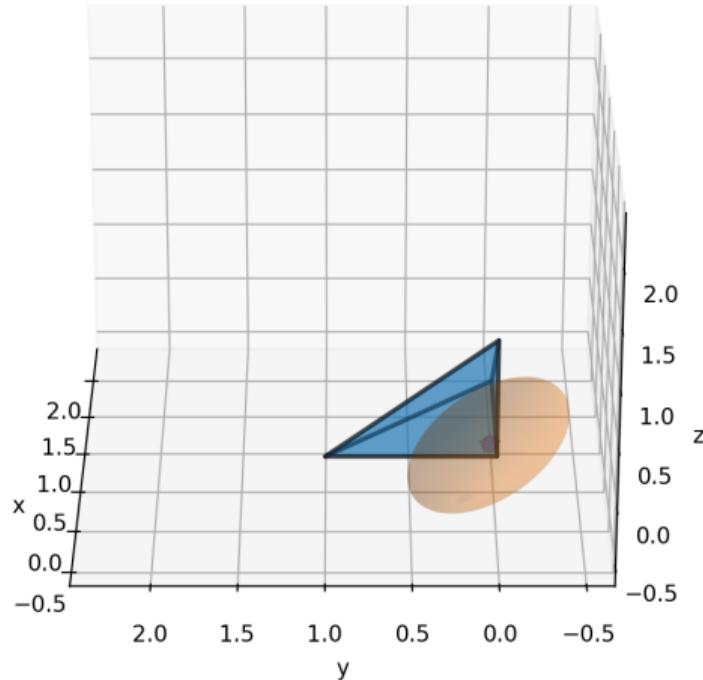
Ellipsoid Method (3D) — iter 29 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 0.951 | best $d^T x \approx 0.205$



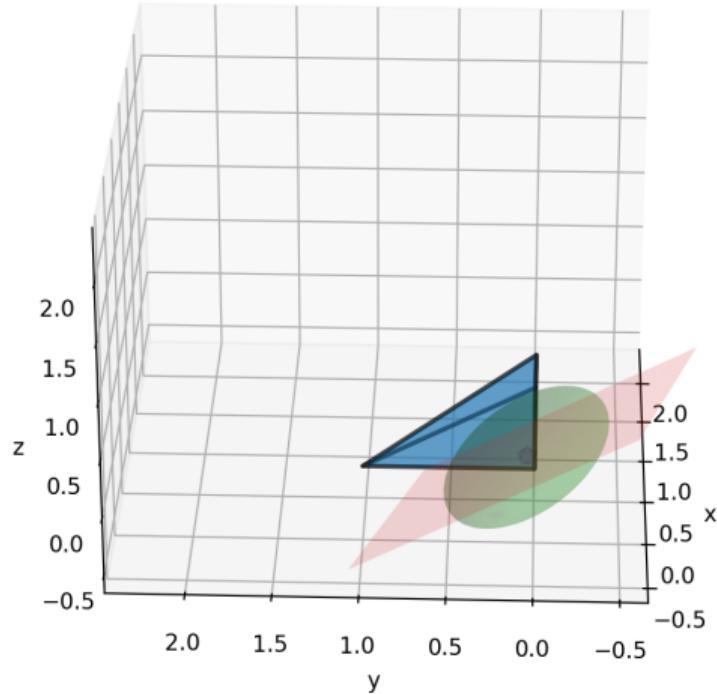
Ellipsoid Method (3D) — iter 29 (show cut plane)
infeasible center | constraint cut | max-axis ≈ 0.951 | best $d^T x \approx 0.205$



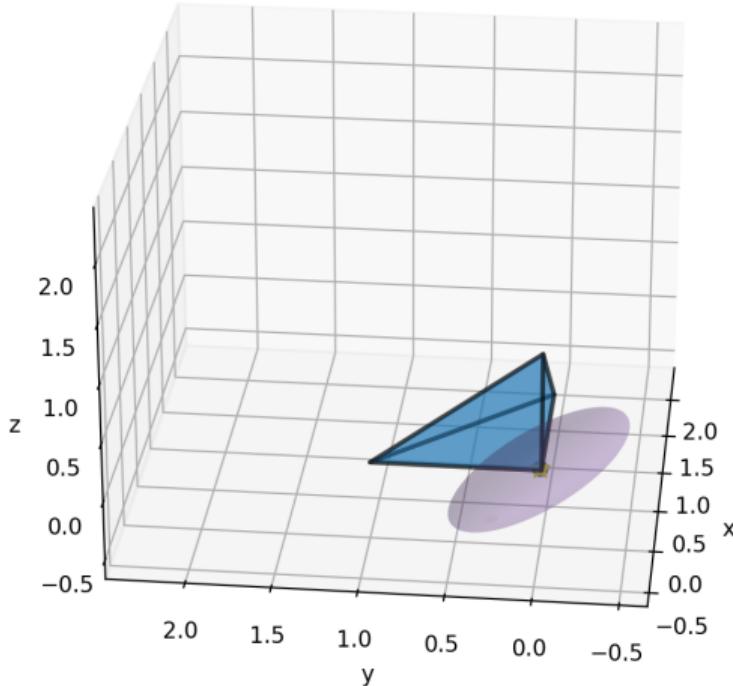
Ellipsoid Method (3D) — iter 30 (show ellipsoid)
feasible center | objective cut | max-axis \approx 0.932 | best $d^T x \approx 0.170$



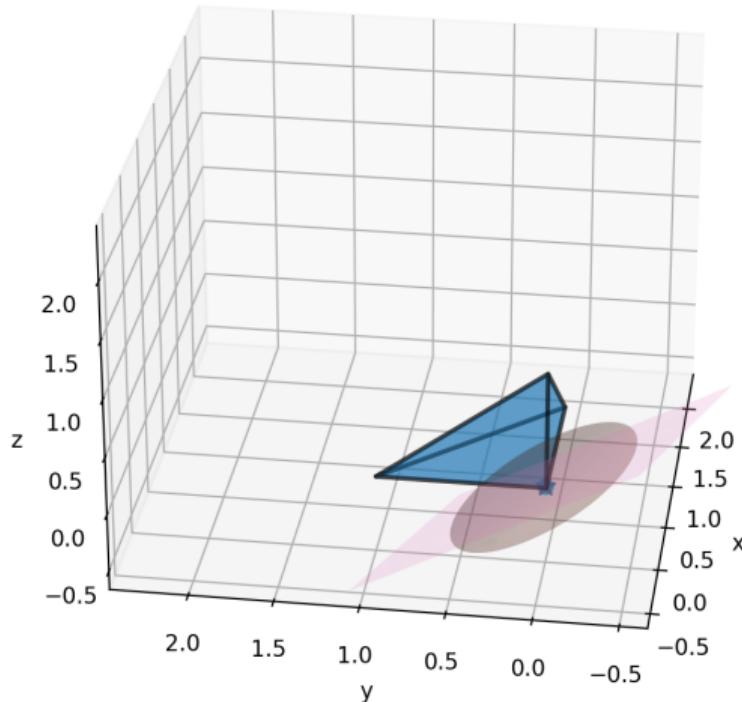
Ellipsoid Method (3D) — iter 30 (show cut plane)
feasible center | objective cut | max-axis ≈ 0.932 | best $d^T x \approx 0.170$



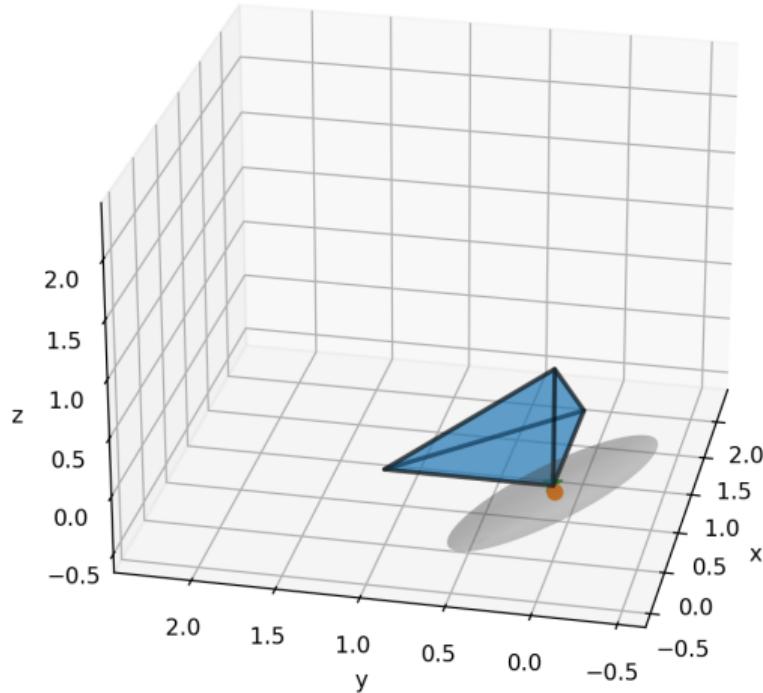
Ellipsoid Method (3D) — iter 31 (show ellipsoid)
feasible center | objective cut | max-axis \approx 0.968 | best $d^T x \approx 0.034$



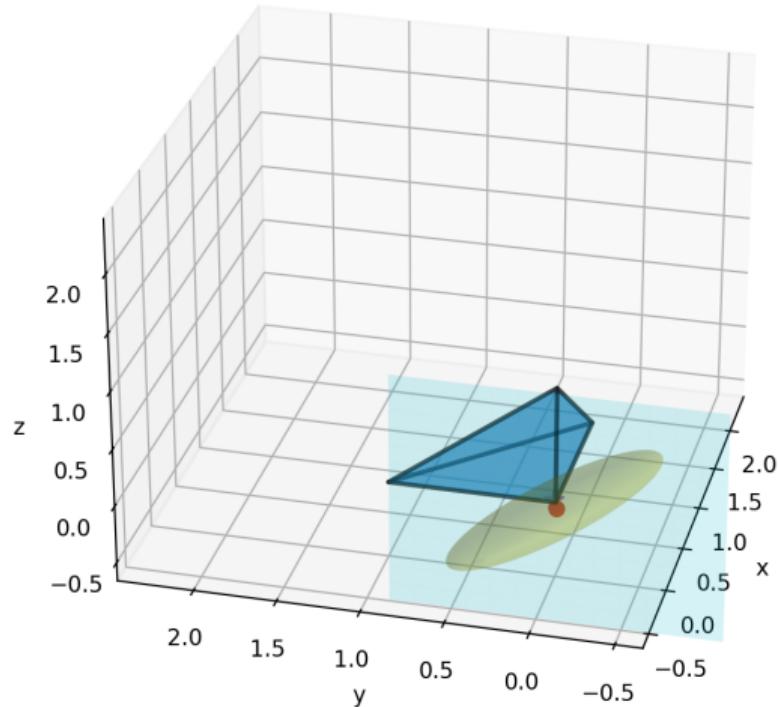
Ellipsoid Method (3D) — iter 31 (show cut plane)
feasible center | objective cut | max-axis \approx 0.968 | best $d^T x \approx 0.034$



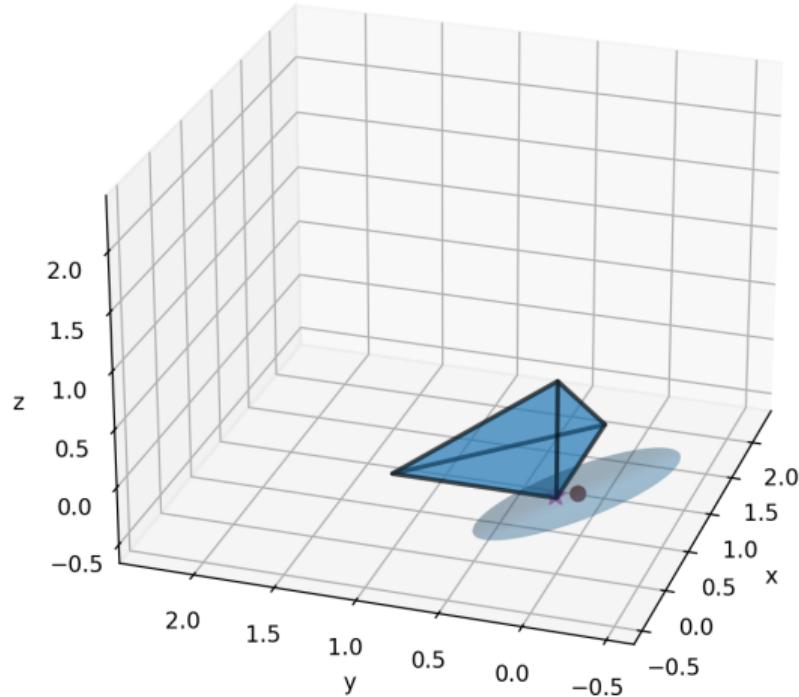
Ellipsoid Method (3D) — iter 32 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 1.016 | best $d^T x \approx 0.034$



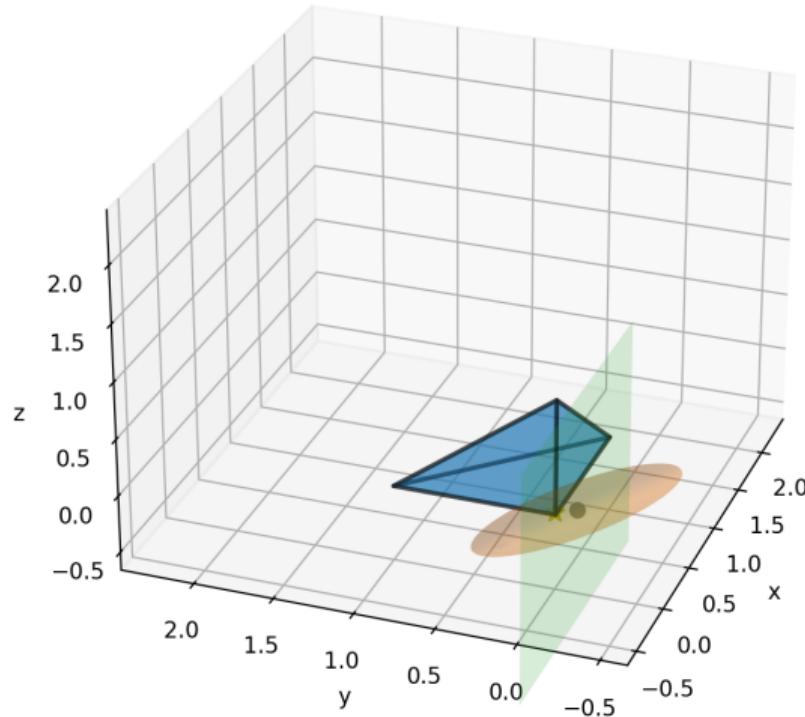
Ellipsoid Method (3D) — iter 32 (show cut plane)
infeasible center | constraint cut | max-axis \approx 1.016 | best $d^T x \approx 0.034$



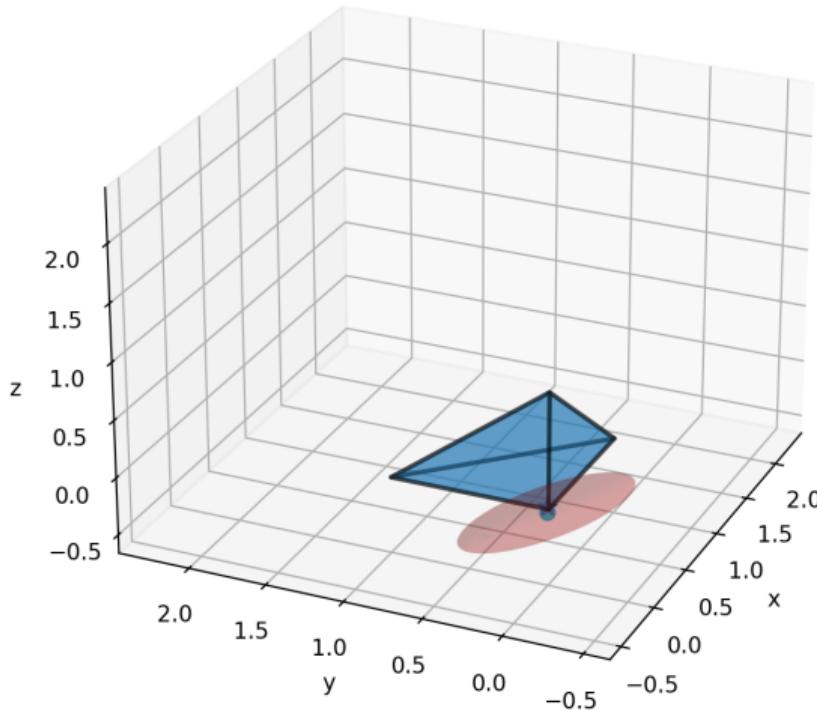
Ellipsoid Method (3D) — iter 33 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 0.764 | best $d^T x \approx 0.034$



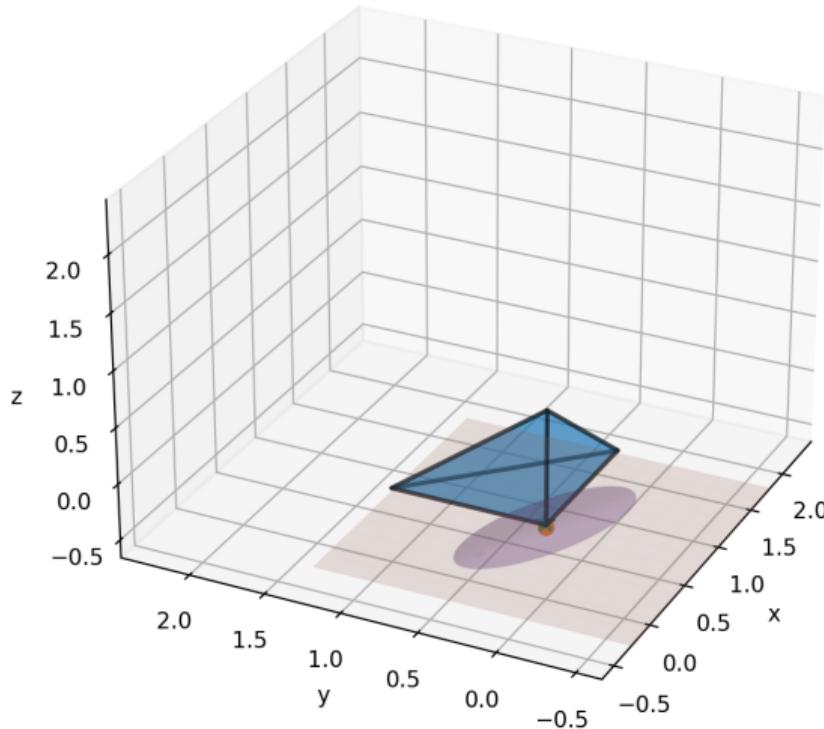
Ellipsoid Method (3D) — iter 33 (show cut plane)
infeasible center | constraint cut | max-axis≈0.764 | best $d^T x \approx 0.034$



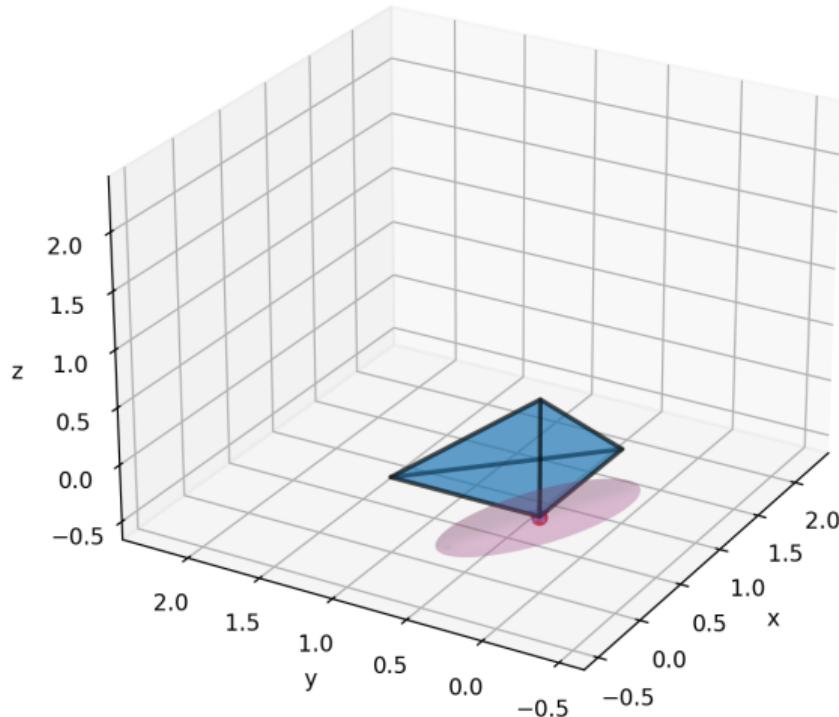
Ellipsoid Method (3D) — iter 34 (show ellipsoid)
infeasible center | constraint cut | max-axis≈0.790 | best $d^T x \approx 0.034$



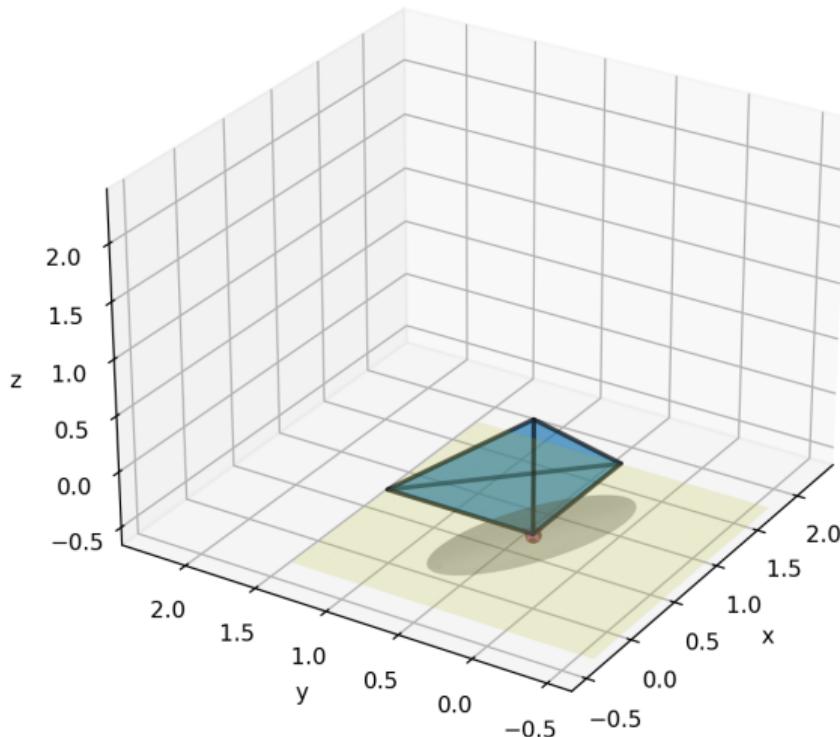
Ellipsoid Method (3D) — iter 34 (show cut plane)
infeasible center | constraint cut | max-axis≈0.790 | best $d^T x \approx 0.034$



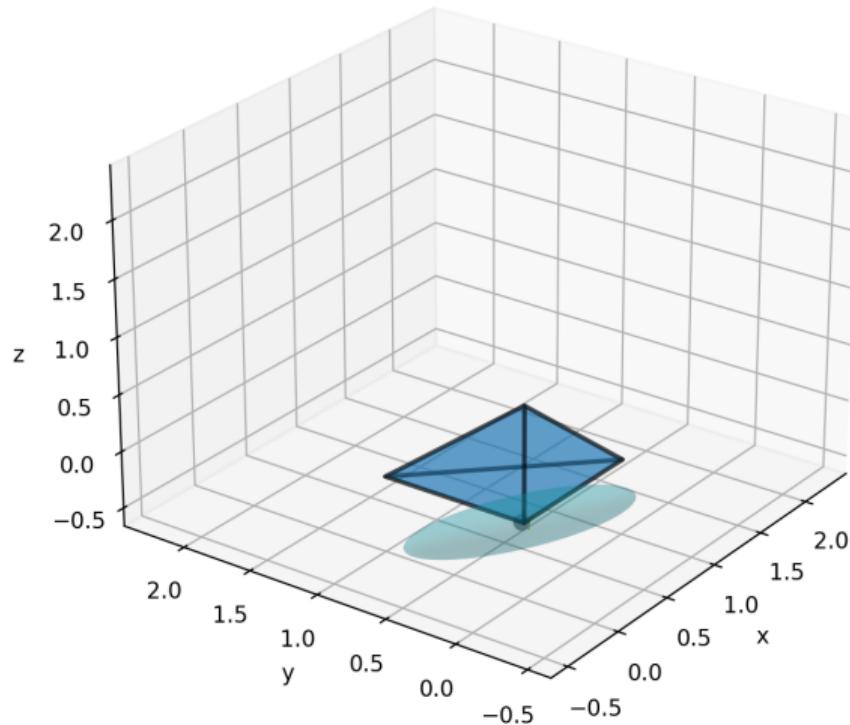
Ellipsoid Method (3D) — iter 35 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 0.764 | best $d^T x \approx 0.034$



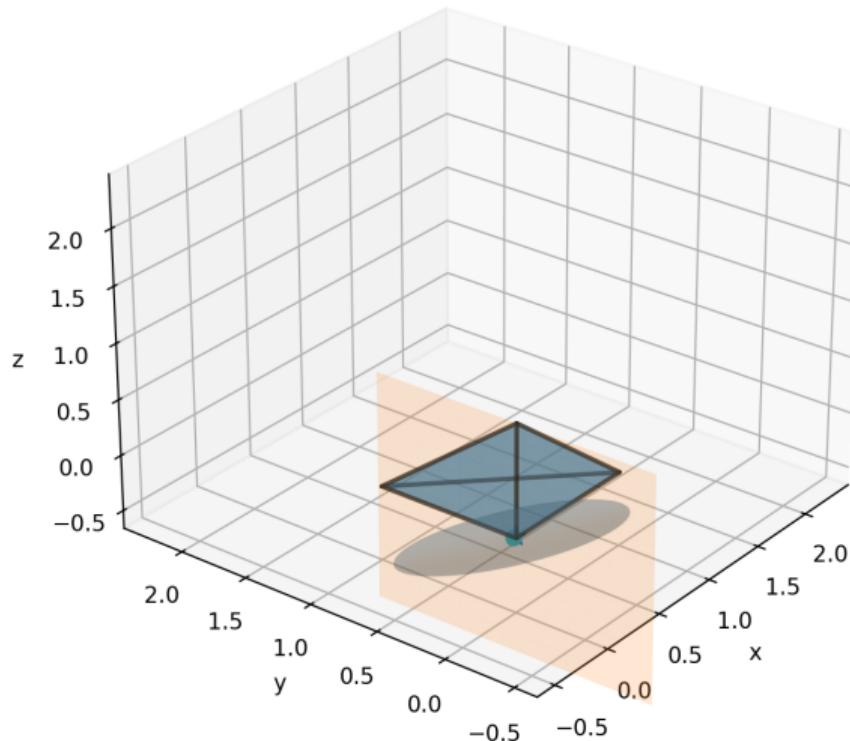
Ellipsoid Method (3D) — iter 35 (show cut plane)
infeasible center | constraint cut | max-axis≈0.764 | best $d^T x \approx 0.034$



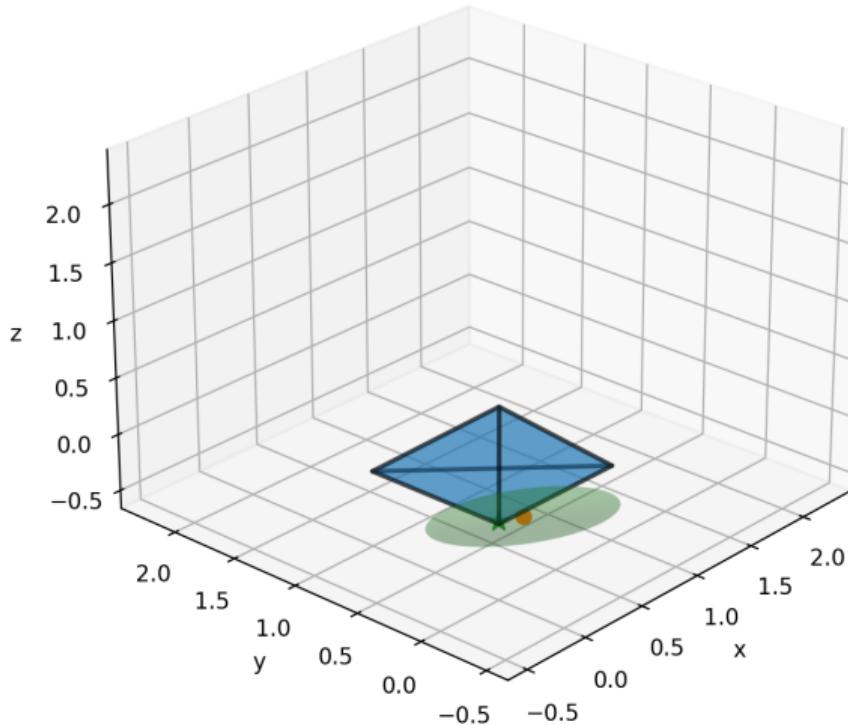
Ellipsoid Method (3D) — iter 36 (show ellipsoid)
infeasible center | constraint cut | max-axis ≈ 0.780 | best $d^T x \approx 0.034$



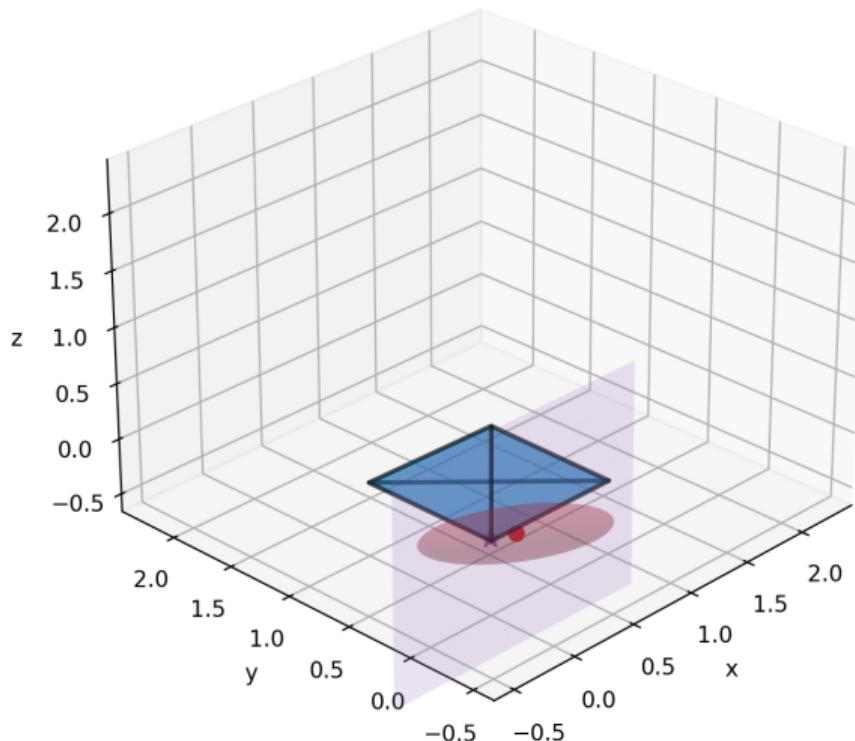
Ellipsoid Method (3D) — iter 36 (show cut plane)
infeasible center | constraint cut | max-axis≈0.780 | best $d^T x \approx 0.034$



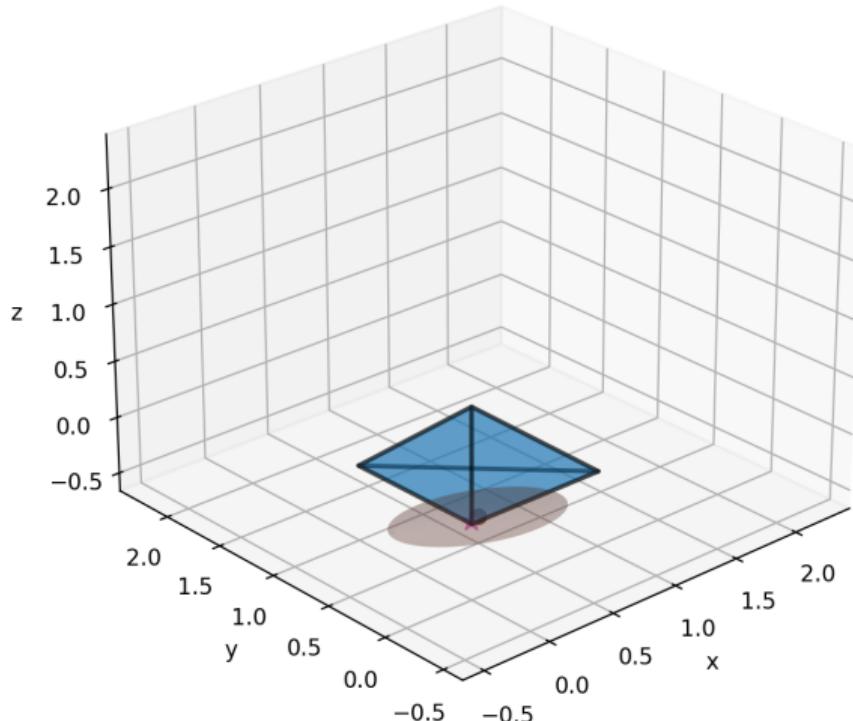
Ellipsoid Method (3D) — iter 37 (show ellipsoid)
infeasible center | constraint cut | max-axis≈0.593 | best $d^T x \approx 0.034$



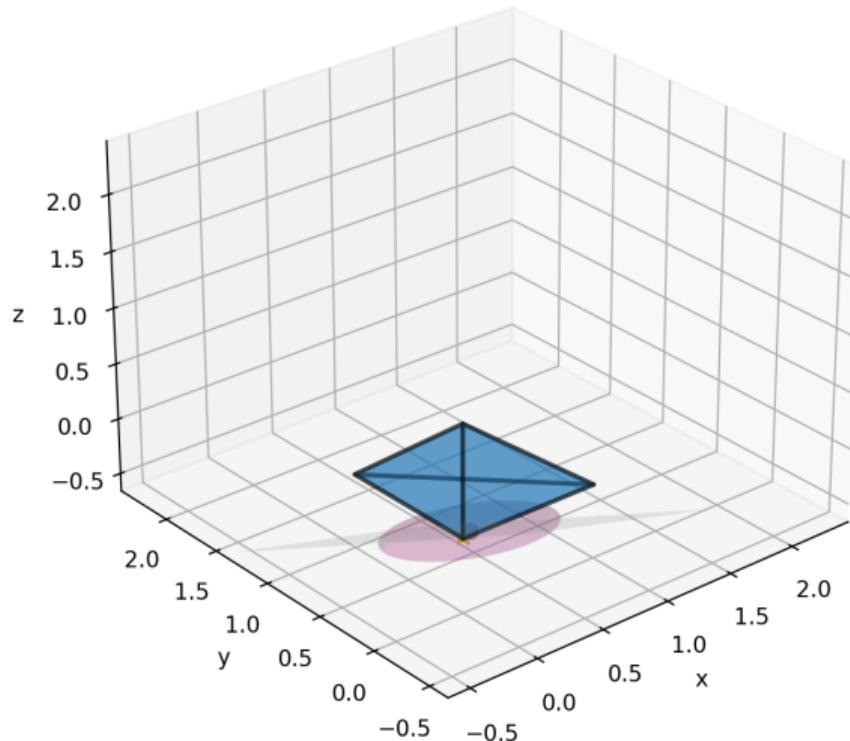
Ellipsoid Method (3D) — iter 37 (show cut plane)
infeasible center | constraint cut | max-axis ≈ 0.593 | best $d^T x \approx 0.034$



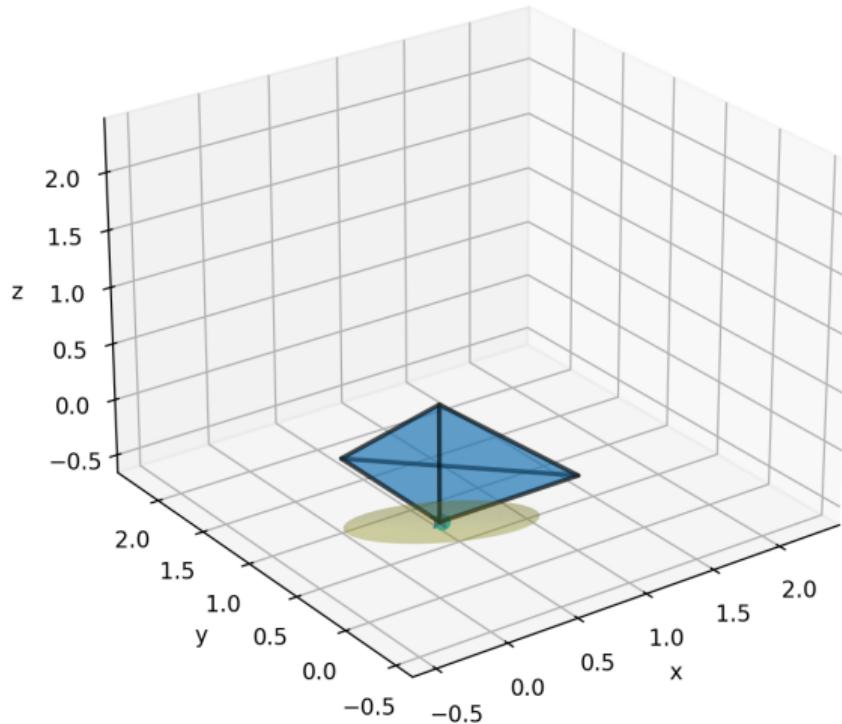
Ellipsoid Method (3D) — iter 38 (show ellipsoid)
feasible center | objective cut | max-axis \approx 0.582 | best $d^T x \approx 0.034$



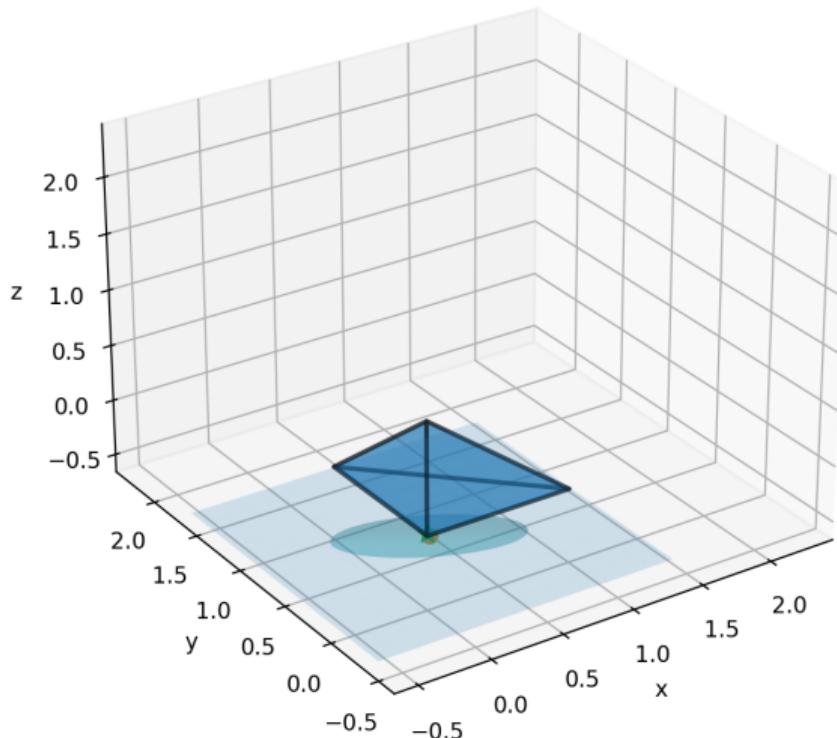
Ellipsoid Method (3D) — iter 38 (show cut plane)
feasible center | objective cut | max-axis \approx 0.582 | best $d^T x \approx 0.034$



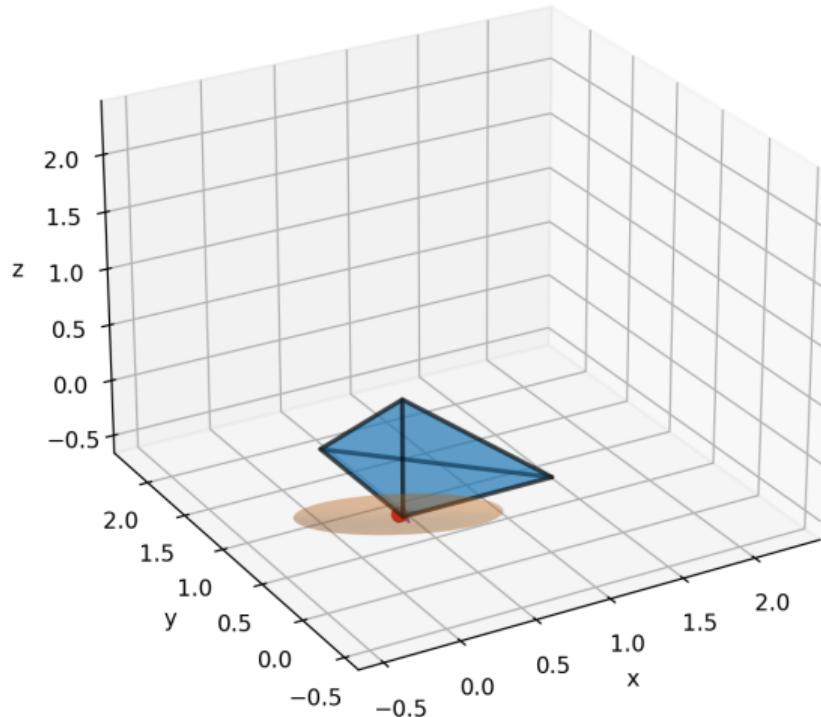
Ellipsoid Method (3D) — iter 39 (show ellipsoid)
infeasible center | constraint cut | max-axis≈0.600 | best $d^T x \approx 0.034$



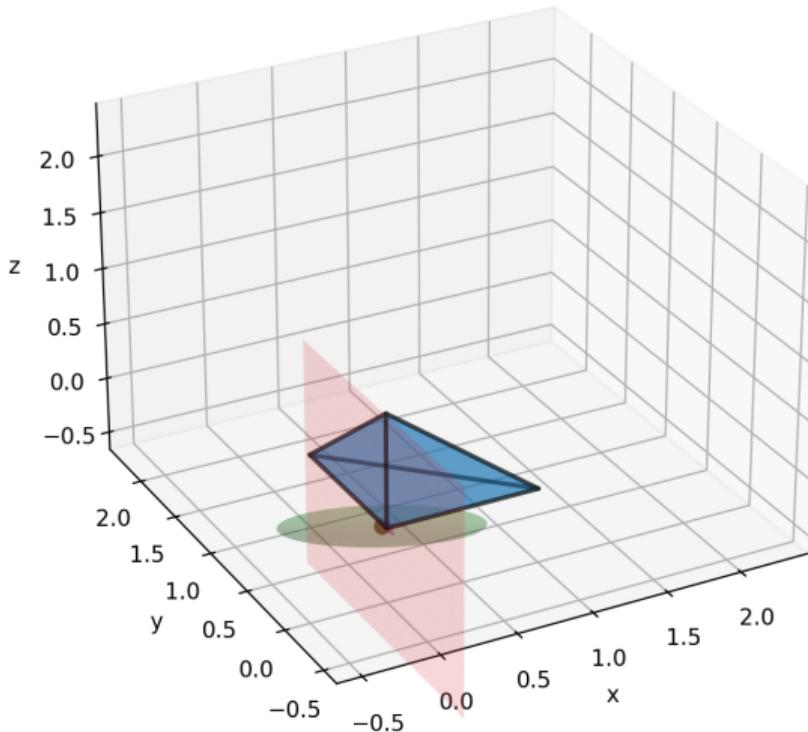
Ellipsoid Method (3D) — iter 39 (show cut plane)
infeasible center | constraint cut | max-axis≈0.600 | best $d^T x \approx 0.034$



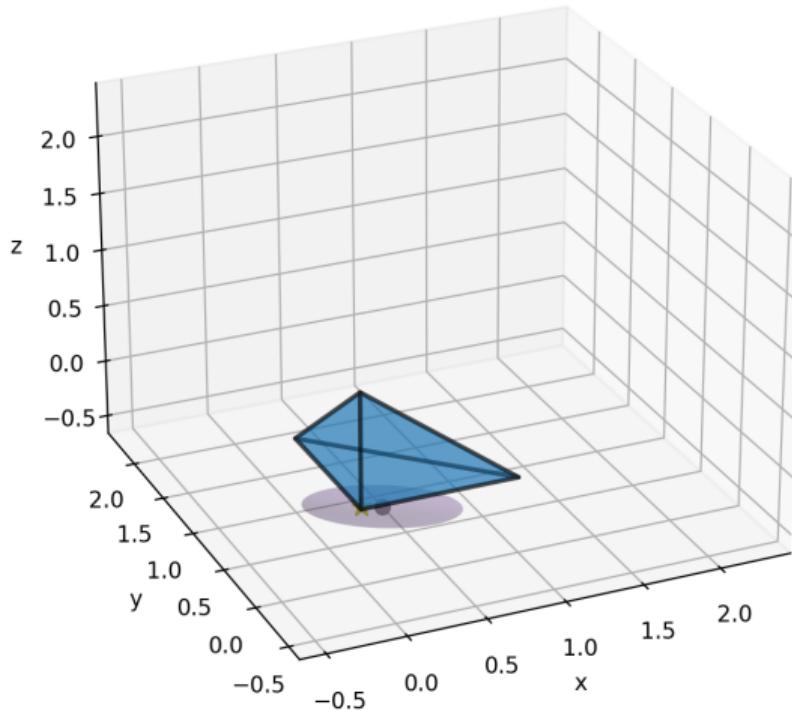
Ellipsoid Method (3D) — iter 40 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 0.612 | best $d^T x \approx 0.034$



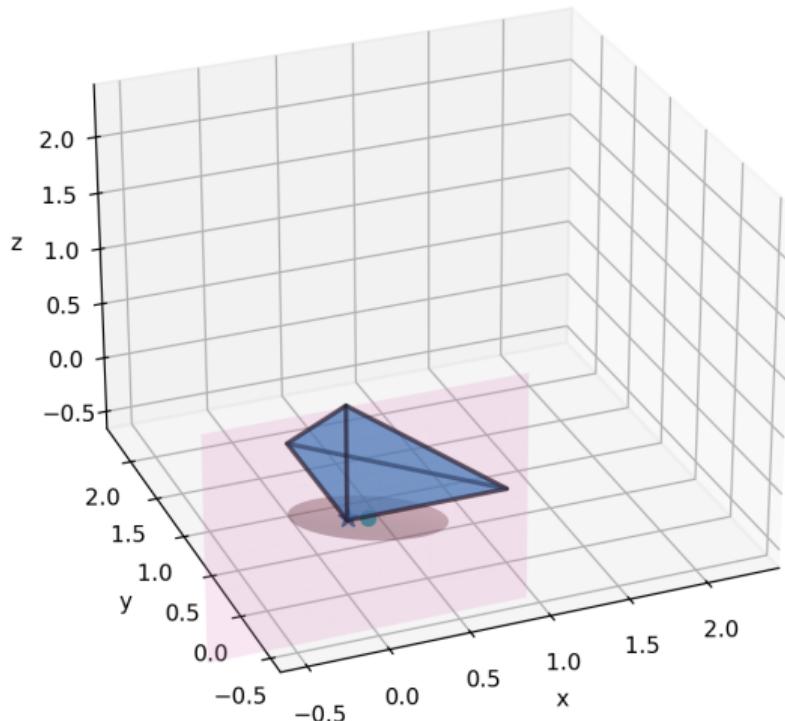
Ellipsoid Method (3D) — iter 40 (show cut plane)
infeasible center | constraint cut | max-axis≈0.612 | best $d^T x \approx 0.034$



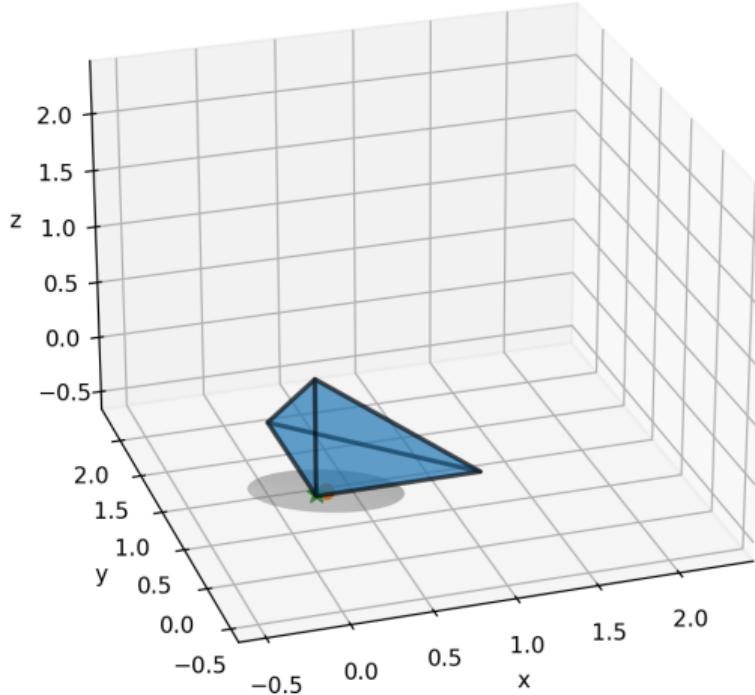
Ellipsoid Method (3D) — iter 41 (show ellipsoid)
infeasible center | constraint cut | max-axis≈0.465 | best $d^T x \approx 0.034$



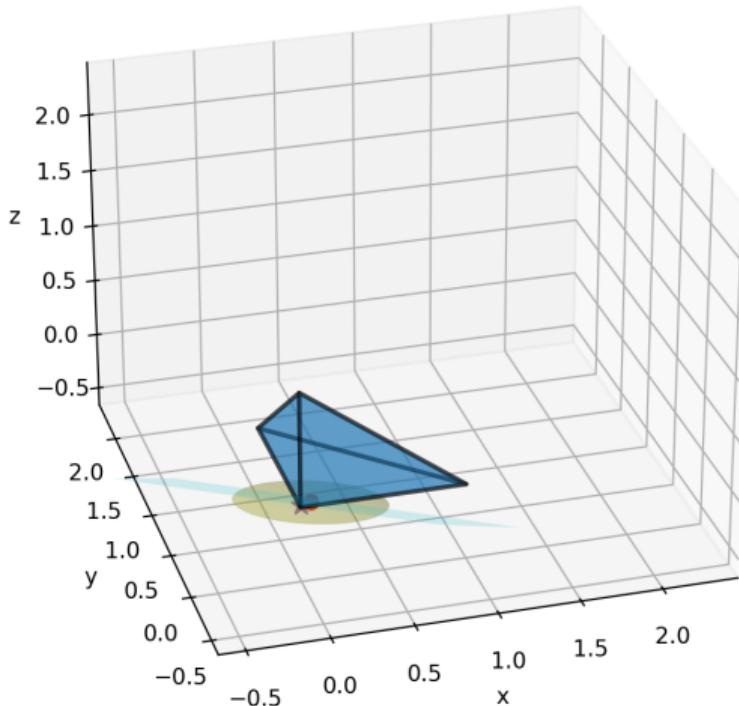
Ellipsoid Method (3D) — iter 41 (show cut plane)
infeasible center | constraint cut | max-axis \approx 0.465 | best $d^T x \approx 0.034$



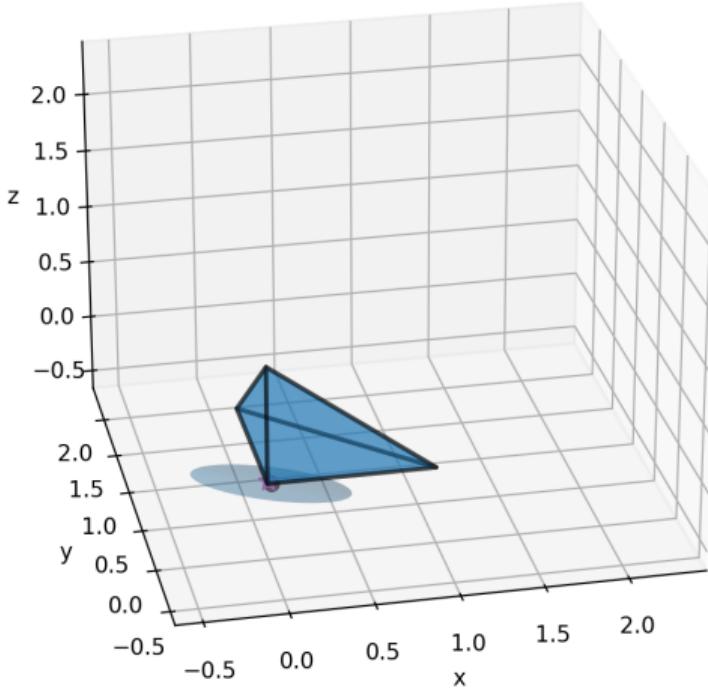
Ellipsoid Method (3D) — iter 42 (show ellipsoid)
feasible center | objective cut | max-axis≈0.459 | best $d^T x \approx 0.034$



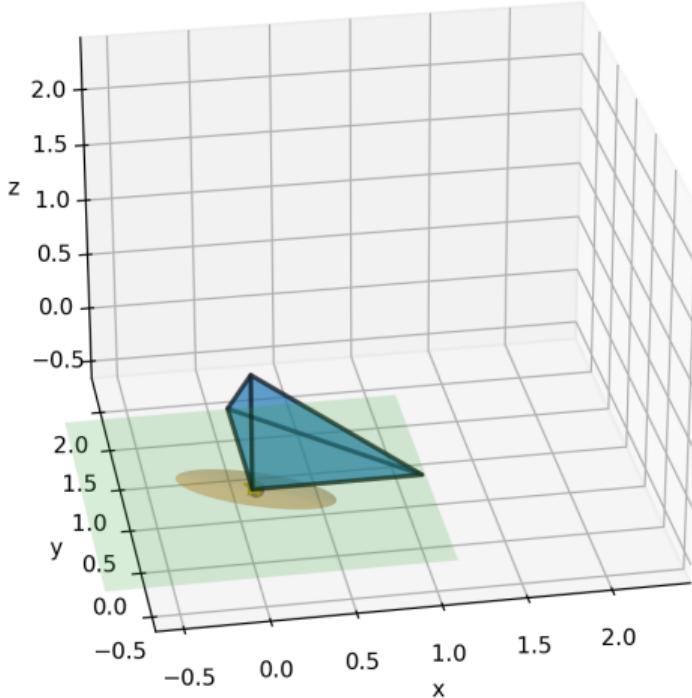
Ellipsoid Method (3D) — iter 42 (show cut plane)
feasible center | objective cut | max-axis \approx 0.459 | best $d^T x \approx 0.034$



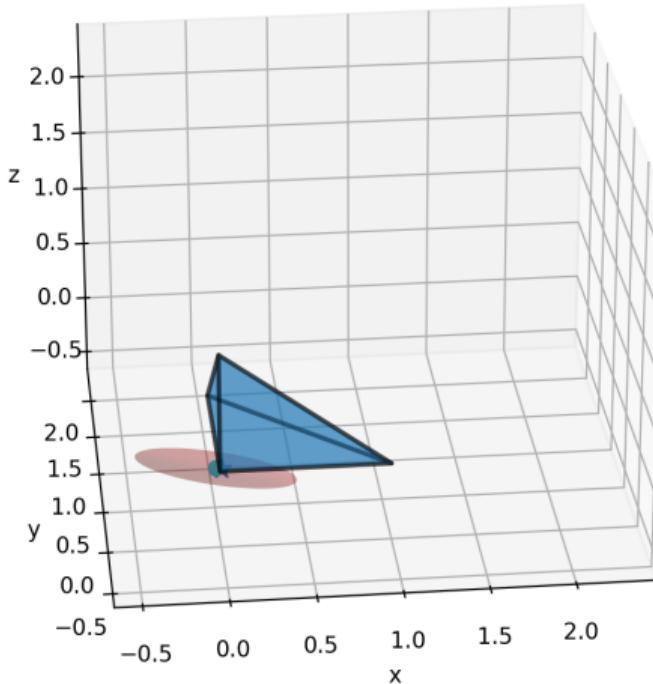
Ellipsoid Method (3D) — iter 43 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 0.474 | best $d^T x \approx 0.034$



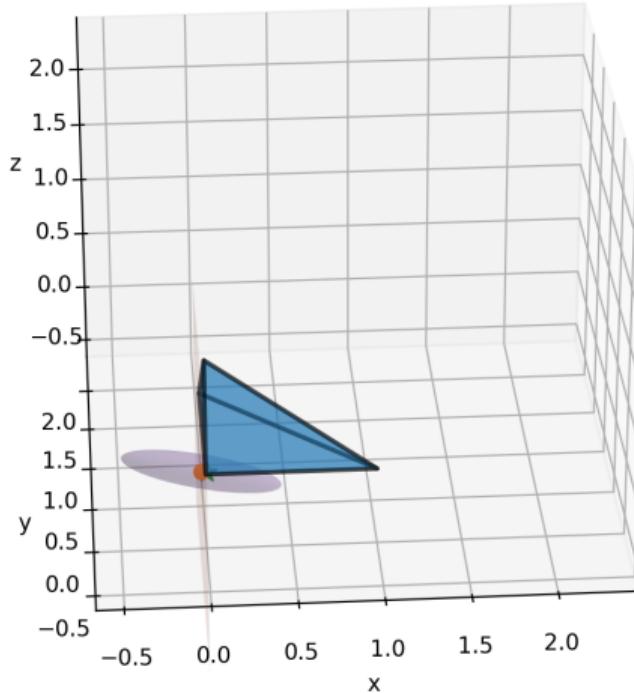
Ellipsoid Method (3D) — iter 43 (show cut plane)
infeasible center | constraint cut | max-axis \approx 0.474 | best $d^T x \approx 0.034$



Ellipsoid Method (3D) — iter 44 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 0.482 | best $d^T x \approx 0.034$



Ellipsoid Method (3D) — iter 44 (show cut plane)
infeasible center | constraint cut | max-axis \approx 0.482 | best $d^T x \approx 0.034$



Ellipsoid Method (3D) — iter 45 (show ellipsoid)
infeasible center | constraint cut | max-axis \approx 0.366 | best $d^T x \approx 0.034$

