An algorithm to find the point with minimum norm in a convex polygon on the centered unit square

The presented method finds among the points that belong to a convex polygon and the centered unit square in the Euclidean plane the one point with minimum distance to the origin (as Figure 1 illustrates). Being very similar to previous work [1], the method sligthly varies the approach of incremental linear programming [2], [3].

Algorithm 2 receives the convex polygon as a set of halfplanes $\left\{\tilde{H}_j\right\}_{j=1}^{m}$ whose intersection defines the polygon. It augments these constraints by the halfplanes $\{H_k^s\}_{k=1}^{4}$ whose intersection defines the centered unit square. To find the solution, it requires to know the solution of the same problem with one constraining halfplane less and which is the respective halfplane. The algorithm allows to incrementally build the final solution for any convex polygon, by solving first the problem without any constraints, then adding one more halfplane and solving again, and so on, until it has processed all the halfplanes. Algorithm 1 implements this approach.

The design of Algorithm 2 allows to avoid numerical problems implied by intersections between (close-to) parallel lines using a simple trick. Namely, one can introduce a small distance tolerance δ to bias every query whether a halfplane contains a point (i.e. every halfplane is enlarged by δ only while performing

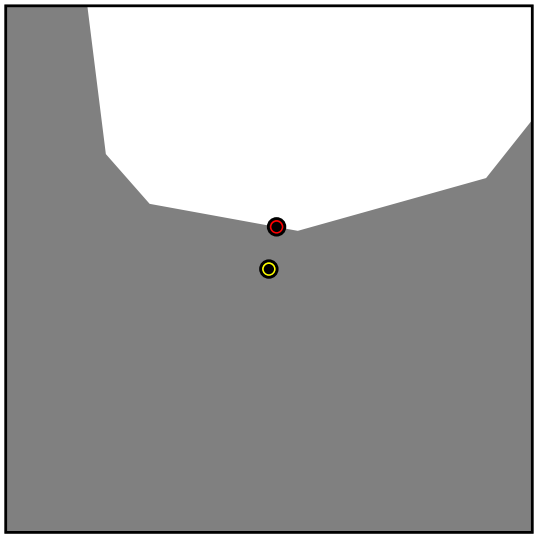

Figure 1: The centered unit square (grey), with side length equal to one, and a superposed convex polygon (white). The points indicate the coordinate origin (yellow-black) and its closest point (red-black) belonging to the convex polygon and the unit square.

Algorithm 1 Incremental Distance Minimization

Input: halfplanes $\{H_i\}_{i=1}^{n}$

Output: point p^* closest to $(0,0)$ while belonging to $\bigcap_{k=1}^{4} H_k^s \bigcap_{i=1}^{n} H_i$

```
p* ← (0,0)
for i ← 1, ..., n do
    p* ← distanceMinimizationIncrementation(H_i, p*, {H_j}_{j=1}^{i-1})
end for
return p*
```

this check). This allows then to introduce a lower threshold α for the angle between two boundary lines which Algorithm 2 attempts to intersect. The algorithm can directly report that the problem is infeasible, whenever an attempt occurs for an angle below α, which needs to be chosen as $\alpha = \sin^{-1}(\delta/\sqrt{2})$. This value for α guarantees that angles below would make the intersection point lie outside the feasible unit square. Thus, the algorithm never has to perform line intersections for very small angles. One can implement Algorithm 2 in a robust way by incorporating this feature, which the pseudo-code omits for the sake of simplicity.

References

[1] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*, pp. 3–19, Springer, 2011.

[2] R. Seidel, "Small-dimensional linear programming and convex hulls made easy," *Discrete & Computational Geometry*, vol. 6, no. 3, pp. 423–434, 1991.

[3] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational geometry: algorithms and applications*. Springer, 2008.

Algorithm 2 Distance Minimization Incrementation

Input: halfplane H, prior solution $\tilde{p}^*$, prior halfplanes $\left\{\tilde{H}_j\right\}_{j=1}^{m}$

Output: point p^* closest to $(0,0)$ while belonging to $\bigcap_{k=1}^{4} H_k^s \bigcap_{j=1}^{m} \tilde{H}_j \cap H$

$p^* \leftarrow \tilde{p}^*$ //*initialize the solution as the prior solution*
if $p^* \notin H$ **then**
  $p^* \leftarrow$ projectOriginOrthogonallyOnHalfplaneBoundary(H)
  $p_1 \leftarrow$ pointShiftedAlongBoundaryBy$(p^*, H, +\sqrt{2})$
  $p_2 \leftarrow$ pointShiftedAlongBoundaryBy$(p^*, H, -\sqrt{2})$
  for $k \leftarrow 1, ..., 4$ **do**
    if $(p_1 \notin H_k^s)\,\&\,(p_2 \in H_k^s)$ **then**
      $p_1 \leftarrow$ intersectHalfplaneBoundaries(H_k^s, H) //*update segment* $[p_1, p_2]$
    else if $(p_1 \in H_k^s)\,\&\,(p_2 \notin H_k^s)$ **then**
      $p_2 \leftarrow$ intersectHalfplaneBoundaries(H_k^s, H) //*update segment* $[p_1, p_2]$
    else if $(p_1 \notin H_k^s)\,\&\,(p_2 \notin H_k^s)$ **then**
      except "infeasible" //*terminate;* p_1*,* p_2 *serve only to determine this*
    end if
    if $p^* \notin H_k^s$ **then**
      $p^* \leftarrow$ intersectHalfplaneBoundaries(H_k^s, H) //*update the solution*
    end if
  end for
  for $j \leftarrow 1, ..., m$ **do**
    if $(p_1 \notin \tilde{H}_j)\,\&\,(p_2 \in \tilde{H}_j)$ **then**
      $p_1 \leftarrow$ intersectHalfplaneBoundaries$(\tilde{H}_j, H)$ //*update segment* $[p_1, p_2]$
    else if $(p_1 \in \tilde{H}_j)\,\&\,(p_2 \notin \tilde{H}_j)$ **then**
      $p_2 \leftarrow$ intersectHalfplaneBoundaries$(\tilde{H}_j, H)$ //*update segment* $[p_1, p_2]$
    else if $(p_1 \notin \tilde{H}_j)\,\&\,(p_2 \notin \tilde{H}_j)$ **then**
      except "infeasible" //*terminate;* p_1*,* p_2 *serve only to determine this*
    end if
    if $p^* \notin \tilde{H}_j$ **then**
      $p^* \leftarrow$ intersectHalfplaneBoundaries$(\tilde{H}_j, H)$ //*update the solution*
    end if
  end for
end if
return p^*