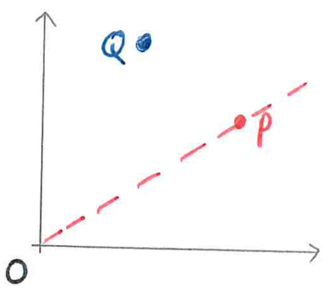
Geometrical algorithms

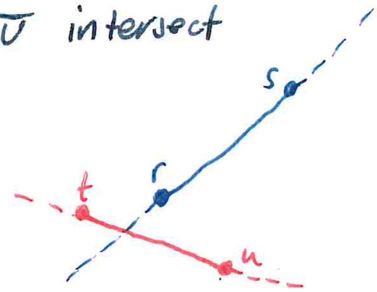
Segment intersection



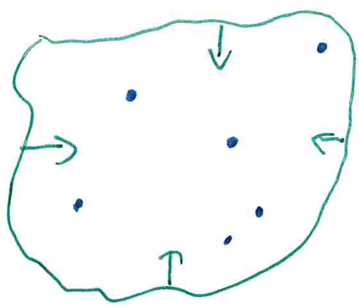
- To find if ~~Q~~ Q is above or below $\vec{OP}$, we rotate $\vec{OP}$ 90° anticlockwise then project $\vec{OQ}$ onto $\vec{OP}$ and check the sign.
- If $p = (p_x, p_y)$ and $q = (q_x, q_y)$, we check the sign of $p^T \cdot q = -p_y q_x + p_x q_y$

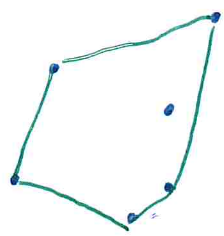
$$p^T \cdot q > 0 \iff Q \text{ above line}$$
$$p^T \cdot q = 0 \iff Q \text{ on line}$$
$$p^T \cdot q < 0 \iff Q \text{ below line.}$$

- This can then be used to decide if $\overline{rs}$ and $\overline{tu}$ intersect
  - if t and u are on the same side of $\vec{rs}$, i.e $(s-r)^T \cdot (t-r)$ and $(s-r)^T \cdot (u-r)$ have the same sign, then they don't intersect
  - if ~~bet~~ r and s are on the same side of $\vec{tu}$, the segments don't intersect
  - else, they intersect.



Convex hull





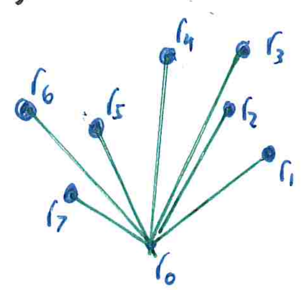
- Tighten a lasso around a set of points.
- Useful for collision detection because all points on an object lie on the same side of one of the line segments on its convex hull.
- Formally, the convex hull is the set of convex combinations, vectors that satisfy $q = \alpha_1 p_1 + \alpha_2 p_2 + \dots + \alpha_n p_n$, $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i = 1$.
- Jarvis's march and Graham's scan can be used to find the corner points of a convex hull.

Jarvis's march

- Start by drawing a horizontal line through the lowest point
- Find the point with the smallest angular separation ↺ and march.
  - ↳ if any points are tied, pick the further one
- Repeat until we return to the original.
- For each point on the convex hull, we had to find the minimum angle for $n-1$ points. ∴ runtime $O(nH)$
- Strategy reminiscent of selection sort.

Graham's scan

- Scan through points in a fan, backtracking when necessary
- Start with the lowest point r_0
- Let $(r_1, r_2, \ldots, r_m)$ be the other points sorted by increasing angle.




```
S = new Stack()
S.push(r1, r2, r3)
for i = 3 to n:
    while ri is not on the left of segment S.first() <-> S.second():
        S.pop()    # backtrack because if we turn right, this point
                   # cannot be on the convex hull
    S.push(pi)
return S.
```

- The loop is $O(n)$ because points cannot be added back to the stack.
- Thus the runtime is $O(n \lg n)$ from the initial sort.