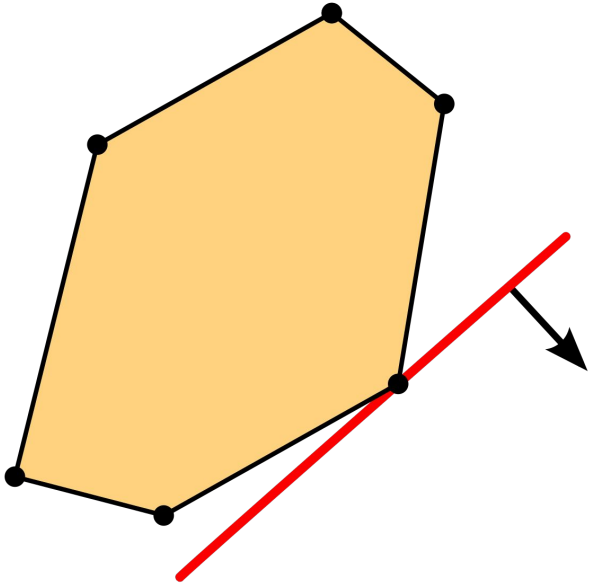
ITMO UNIVERSITY

Linear programming algorithms



Juan camilo Diosa Echeverri - C4134
Kiseleva Anastasiia - C4133
Turukhanov Egor - C4134

ITMO more than a UNIVERSITY

Table of Content

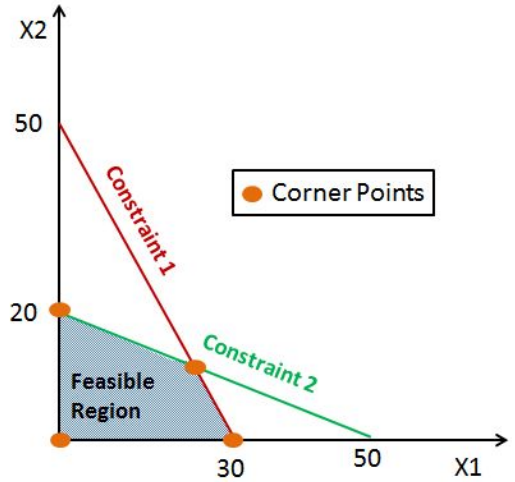
ITMO more than a UNIVERSITY

Definition of Linear Programming Problem

A linear programming algorithm is defined as an algorithm that allows solving a problem of optimization (maximization or minimization) of a linear function subject to linear constraints by applying several processes or steps. These constraints may be equalities or designations.

$$\text{Max } c^T x$$

$$\text{Subject to } Ax \leq b.$$



Simplex Method

This is the standard technique in "Linear Programming" to solve an optimization problem, in which you have an "objective function" and several functions expressed as inequalities called "constraints".

In 1947, George B. Dantzig developed a technique to solve linear programs, inspired by the input output Matrix created by the nobel prize in economics Wassily Leontief.

$$max(min)z = \quad c^T x$$

Subject to:

$Ax=b$

$x \geq 0$

The simplex algorithm is an iterative process that starts from the origin of the n-D vector space $x1=\cdots=xn=0$, and goes through a sequence of vertices of the polytope to eventually arrive at the optimal vertex at which the objective function is maximized

Step 1

$$\max Z = 4x_1 + 3x_2 \ ^{6}$$

$$2x_1 + 3x_2 \leq 6$$

$$-3x_1 + 2x_2 \leq 3$$

$$2x_2 \leq 5$$

$$2x_1 + x_2 \leq 4$$

$$x_1, x_2 \geq 0$$

First convert the standard form of the problem into a tableau

X_1	X_2	b
2	3	6
-3	2	3
0	2	5
2	1	4
-4	-3	0

Step 2

X_1	X_2	b
2	3	6
-3	2	3
0	2	5
2	1	4
-4	-3	0

Add dummy variables for each of the constraints. In this case X_3, X_4, X_5 and X_6

X_1	X_2	X_3	X_4	X_5	X_6	b
2	3	1	0	0	0	6
-3	2	0	1	0	0	3
0	2	0	0	1	0	5
2	1	0	0	0	1	4
-4	-3	0	0	0	0	0

Step 3

We select x_j in the "j" column of the tableau if it is most heavily weighted by $(Cj = \max\{C_1, \ldots, C_n\}$ (-Cj is most negative), as this x_j will increase $z = \sum_{j=1}^{n} c_j x_j$ more than any other x_k $(k \neq j)$.

Choose the constraint with least value $\frac{b_k}{a_{kj}}$.

X_1	X_2	X_3	X_4	X_5	X_6	b
2	3	1	0	0	0	6/2
-3	2	0	1	0	0	-
0	2	0	0	1	0	-
2	1	0	0	0	1	2/2
-4	-3	0	0	0	0	0

Step 4

Divide all elements in the "i" row of A by a_{kj}, so that a_{kj}=1.

Subtract a_{kj}, times the "i" row from the "k" row so that $a_{kj} = 0, (k = 1, \ldots, m, k \neq j)$.

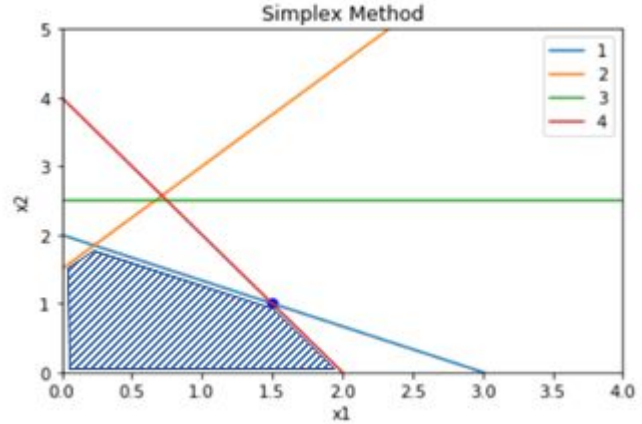
X_1	X_2	X_3	X_4	X_5	X_6	b
0	2	1	0	0	-1	2
0	3.5	0	1	0	1.5	9
0	2	0	0	1	0	5
1	1/2	0	0	0	1/2	2
0	-1	0	0	0	2	8

So a_{kj} becomes 1 while all other elements in the "j" column become zero. This is Gaussian elimination based on pivoting.

Step 5

Repeat the steps above until all elements in the last row are non-negative. In this way the optimal feasible basic solution is found.

X1	X2	X3	X4	X5	X6	b
0	1	0.5	0	0	-0.5	1
0	0	-1.75	1	0	3.25	5.5
0	0	-1	0	1	1	3
1	0	-0.25	0	0	0.75	1.5
0	0	0.5	0	0	1.5	9

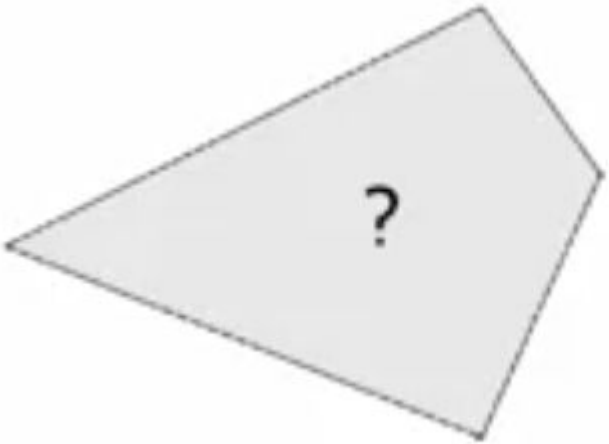


We can use the scipy.optimize library in Python and import linprog to work with the “simplex method”

ITMOre than a UNIVERSITY

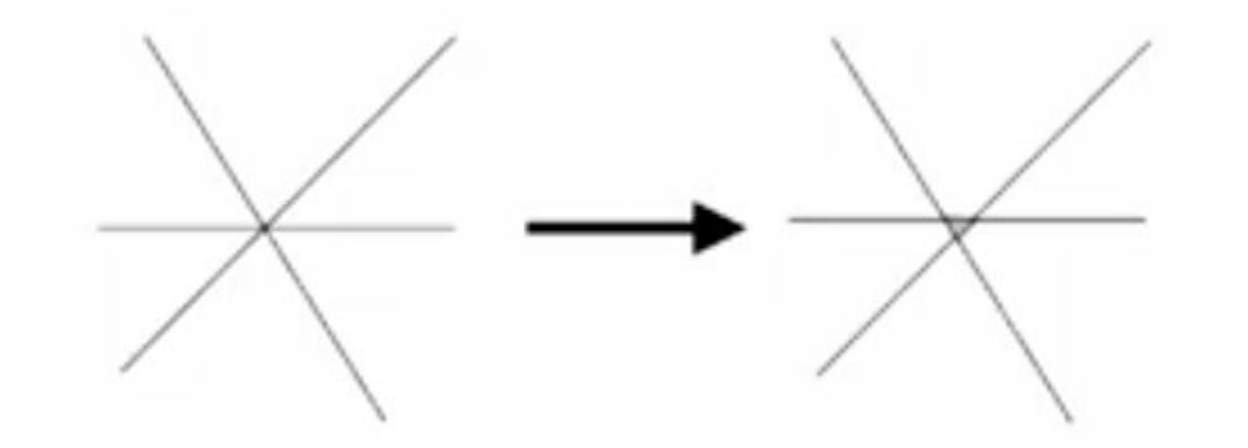
Ellipsoid algorithm

Ellipsoid solves the satisfiability version of a linear program



Step 1

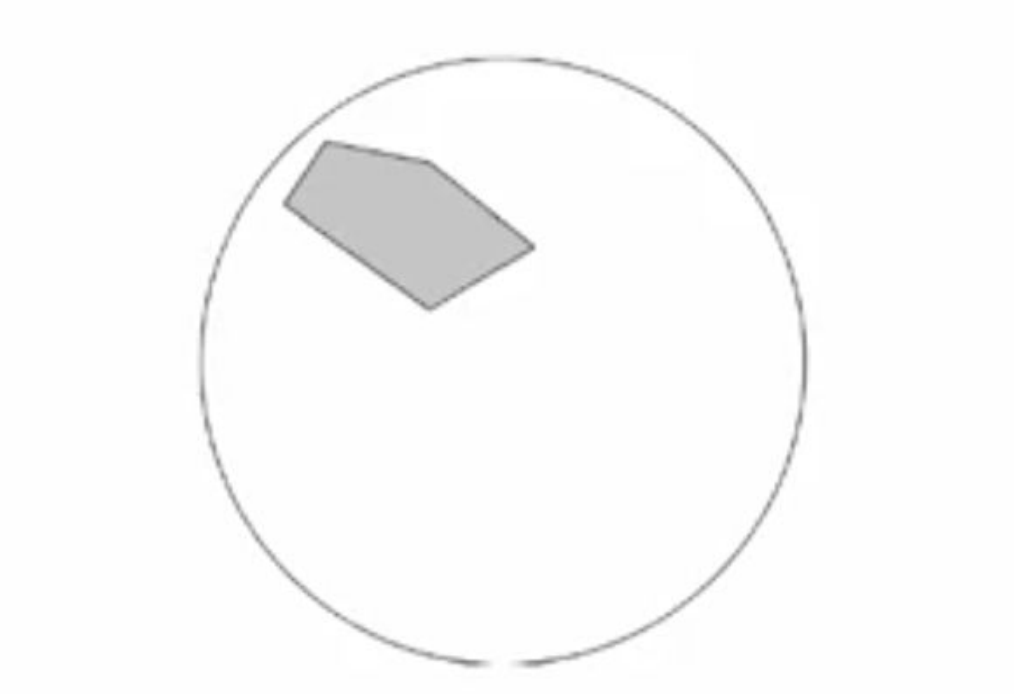
Relax all equations a tiny bit



Solution set

Step 2

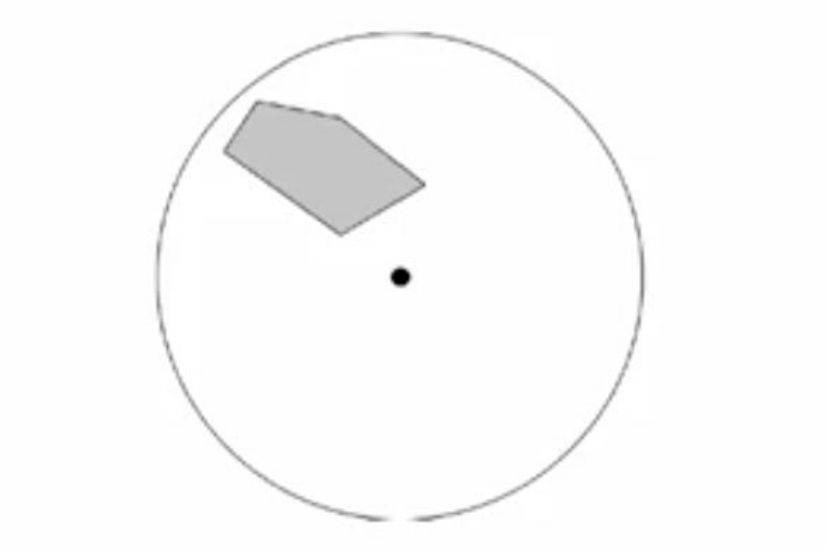
Bound solution set in a large ball



ITMOre than a UNIVERSITY

Step 3

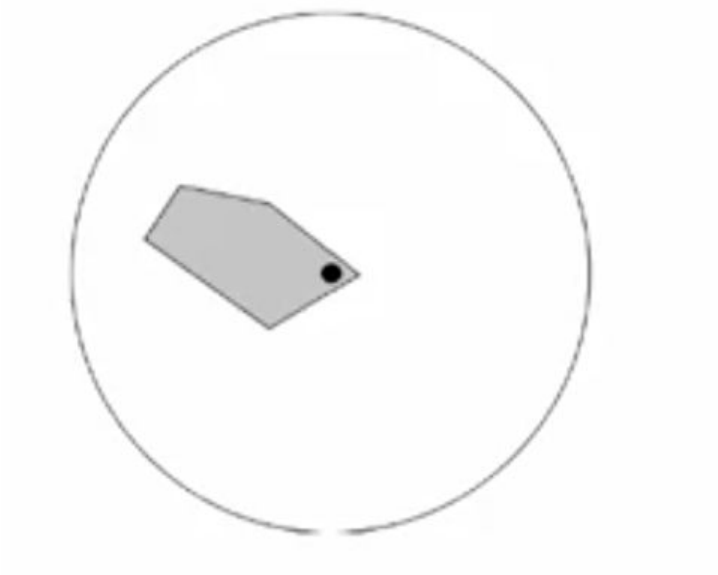
Have ellipsoid that contains all solutions. See if center of ellipsoid is a solution



ITMO more than a UNIVERSITY

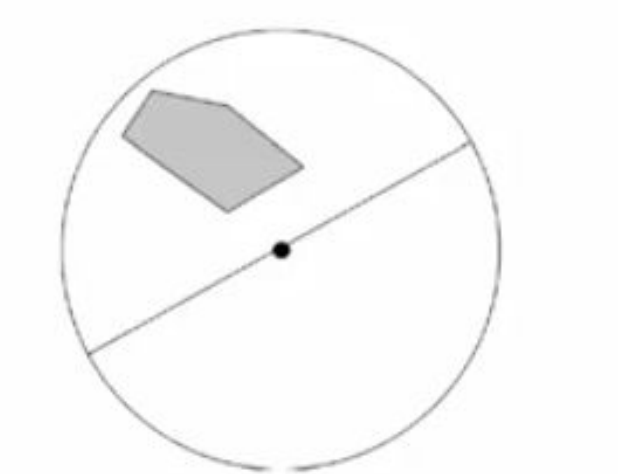
Step 3(a)

If it is a solution, system is satisfiable



Step 3 (b)

If not solution, find separating hyperplane

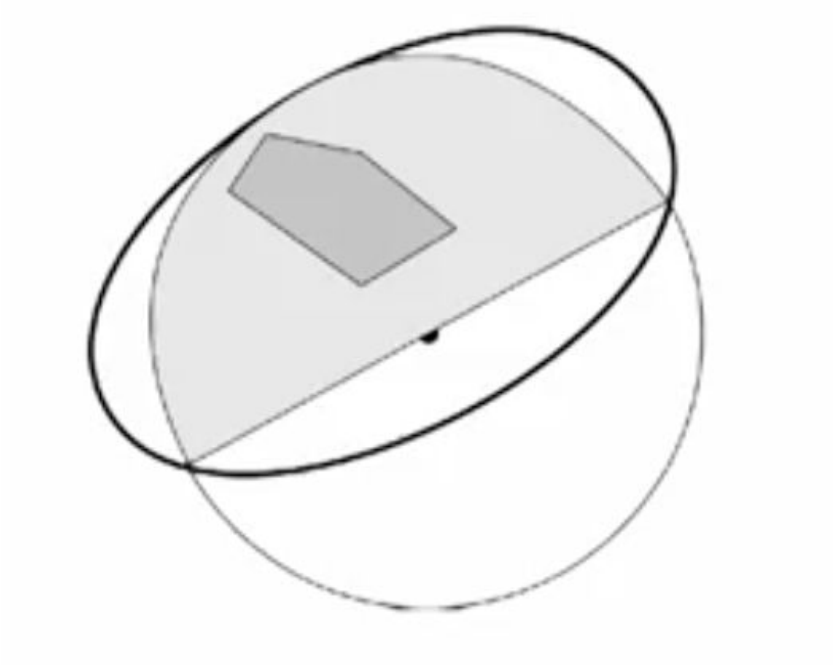


Solution now contained in half-ellipsoid

ITMO more than a UNIVERSITY

Step 3(b)

Half-ellipsoid contained in new ellipsoid of smaller volume



Step 4

Repeat until ellipsoid is too small to contain solution set. In this case, there must be no solutions.

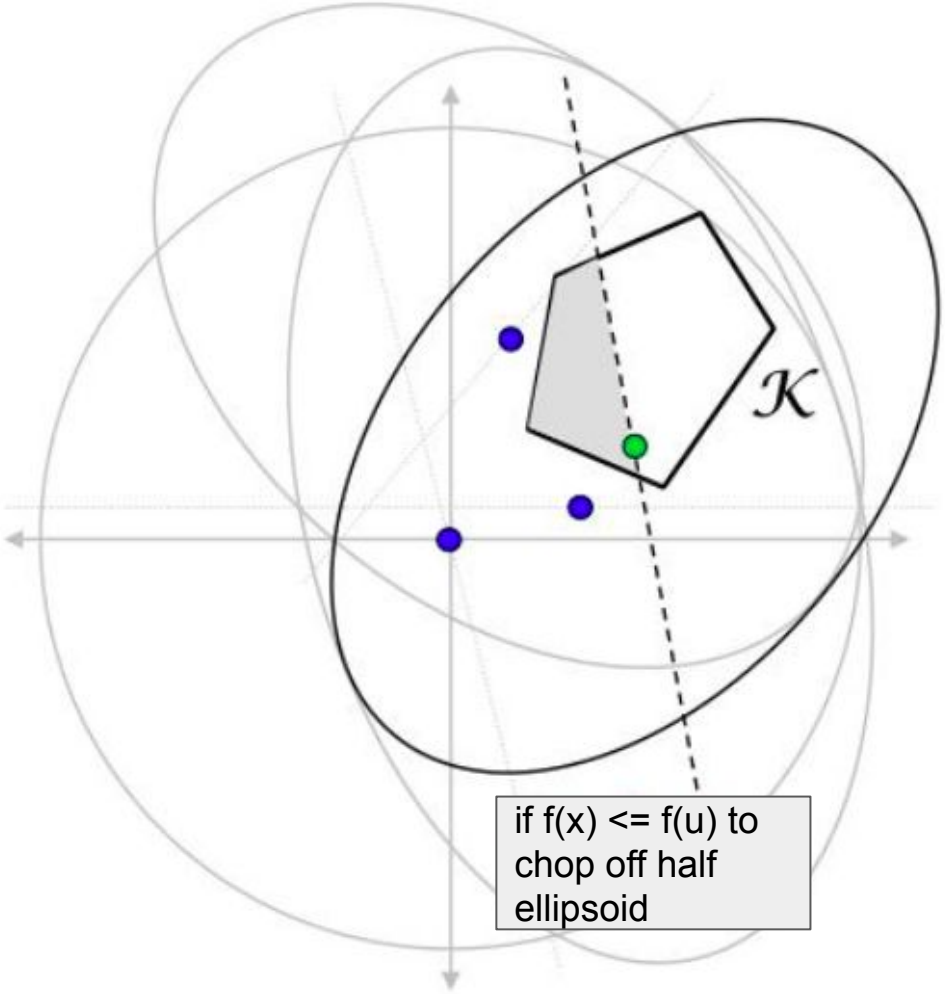
Pseudo-code

1. Start with $k = 0$, $E_0 = E(a_0, A_0) \supseteq P$, $P = \{x : Cx \leq d\}$.
2. While $a_k \notin P$ do:
   - Let $c^T x \leq d$ be an inequality that is valid for all $x \in P$ but $c^T a_k > d$.
   - Let $b = \frac{A_k c}{\sqrt{c^T A_k c}}$.
   - Let $a_{k+1} = a_k - \frac{1}{n+1} b$.
   - Let $A_{k+1} = \frac{n^2}{n^2-1}(A_k - \frac{2}{n+1} b b^T)$.

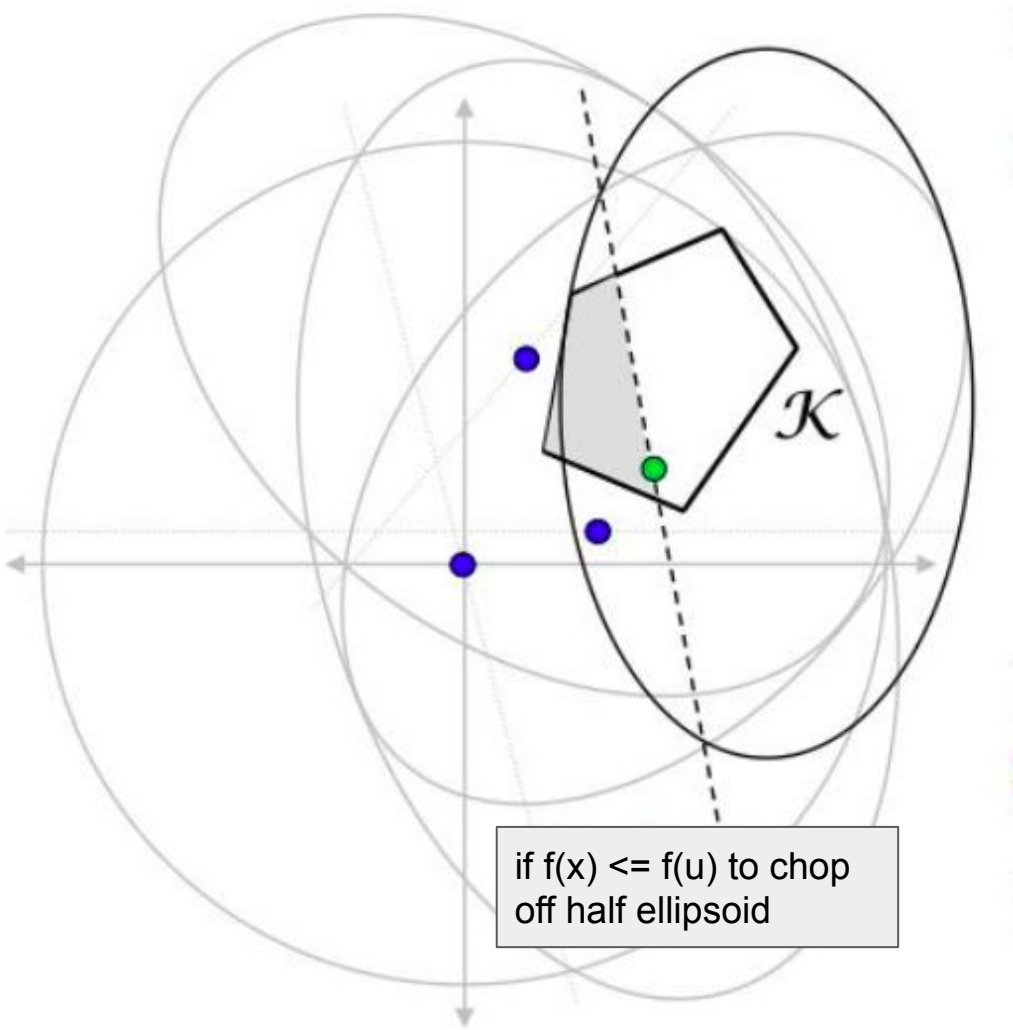
Application to linear programming

1. If in the linear programming problem it was possible to construct a ball containing the desired solution, then it can be solved by the ellipsoid method.
2. Find some point u inside the ball, satisfying the constraints of the problem. (previous steps)
3. Draw a hyperplane f(x) < f(u), where f - objective function
4. Find a point at the intersection of the original and new hyperplane
5. With new found point we do the same

ITMO more than a UNIVERSITY



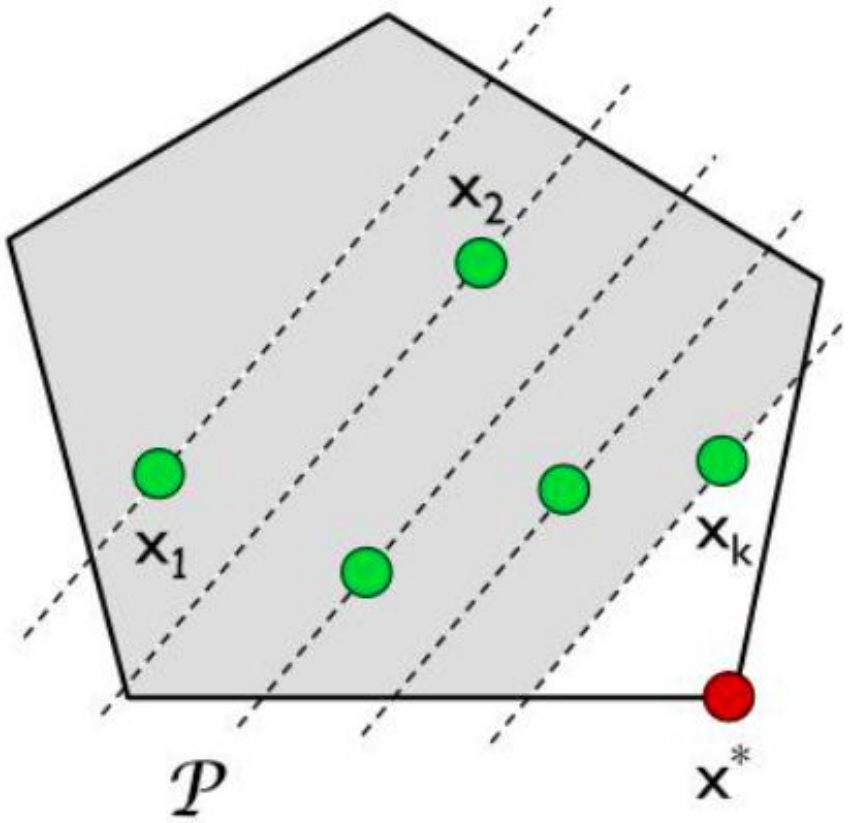
ITMO more than a UNIVERSITY



if f(x) <= f(u) to chop off half ellipsoid

ITMO more than a UNIVERSITY

https://hsto.org/webt/np/p9/6n/npp96n1_sw560-19laylj5big5s.gif



Ellipsoid ≡ squashed sphere

Start with ball containing (polytope) $\mathcal{K}$.

y_i = center of current ellipsoid.

If $y_i \in \mathcal{K}$, use objective function cut $c \cdot x \geq c \cdot y_i$ to chop off $\mathcal{K}$, half-ellipsoid.

If $y_i \notin \mathcal{K}$, use separating hyperplane to chop off infeasible half-ellipsoid.

New ellipsoid = min. volume ellipsoid containing "unchopped" half-ellipsoid.

Repeat for $i=0,1,\ldots,T$.

Runtime

Ellipsoid runs in polynomial time

$O((m+n^2)\, n^5 \log(nU))$

where

n = Dimension

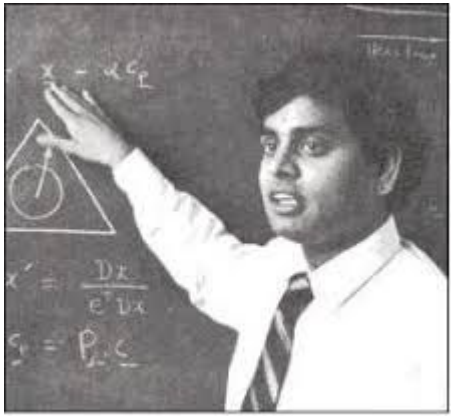
m = Number of Constraints

U = Numerical Size of Coefficients

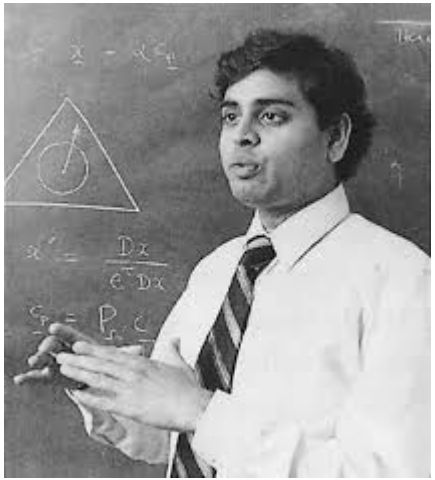
ITMO more than a UNIVERSITY

Projective Algorithm

Narendra Karmarkar - 1984



Narendra Karmarkar



Alternative version - Affine Scaling algorithm (1985) - more intuitive version.

Runtime of Projective Algorithm: $O(n^{3,5}L^2)$

L - number of bits.

Algorithm formulation

Step 1. Define a scaling matrix $D = \begin{pmatrix} x_1 & 0 & 0 \\ 0 & x_2 & 0 \\ 0 & 0 & x_3 \end{pmatrix}$ and $D^{-1} = \begin{pmatrix} \frac{1}{x_1} & 0 & 0 \\ 0 & \frac{1}{x_2} & 0 \\ 0 & 0 & \frac{1}{x_3} \end{pmatrix}$.

Compute $\tilde{A} = AD$
$\tilde{c} = D\vec{c}$

For n = 3

Step 2. Compute $P = I - \tilde{A}^T \left(\tilde{A}\tilde{A}^T\right)^{-1} \tilde{A}.$ and $\tilde{c}_p = P\tilde{c}$

Step 3. Compute $\tilde{x}_{\text{new}} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + k\tilde{c}_p$ k > 0.

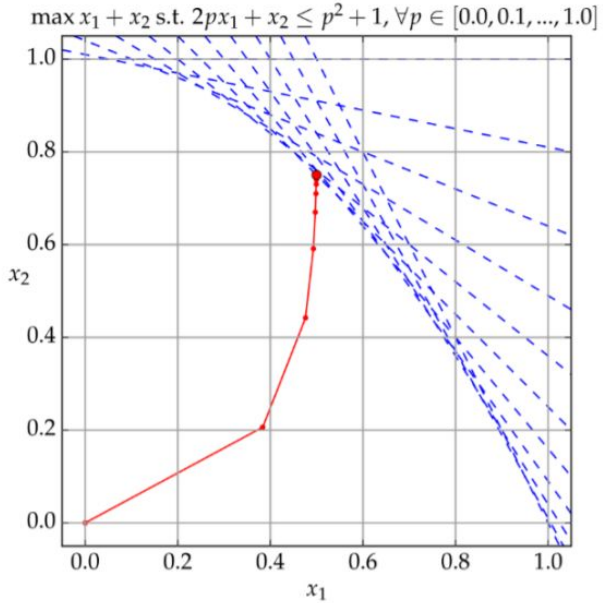
if we take ½ between were we are and edge of feasible set. Then, $k = \frac{-\frac{1}{2}}{\min\{\tilde{c}_p\}}$

Step 4. Convert to x to original form by $\vec{x}_{\text{new}} = D\tilde{x}_{\text{new}}$

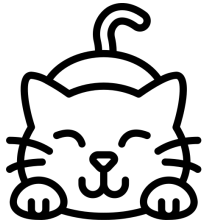
Check for feasibility and optimality.

Simpler algorithm same trace

Gill 1985 - Newton Barrier Method - captures the form of modern interior point methods.



Practical problem



We want to produce cat food products as cheaply as possible while ensuring they meet the stated nutritional analysis requirements shown on the cans.

Stuff	Protein	Fat	Fibre	Salt
Chicken	0.100	0.080	0.001	0.002
Beef	0.200	0.100	0.005	0.005
Rice	0.000	0.010	0.100	0.002
Wheat bran	0.040	0.010	0.150	0.008

Assume we have only **Chicken** and **Beef**. Then our **decision variables are**:

$x_1 =$ percentage of chicken meat in a can of cat food

$x_2 =$ percentage of beef used in a can of cat food

and **objective function** is: $\mathbf{min}\ 0.013x_1 + 0.008x_2$

Constraints for our problem

$$1.000x_1 + 1.000x_2 = 100.0$$
$$0.100x_1 + 0.200x_2 \geq 8.0$$
$$0.080x_1 + 0.100x_2 \geq 6.0$$
$$0.001x_1 + 0.005x_2 \leq 2.0$$
$$0.002x_1 + 0.005x_2 \leq 0.4$$

Algorithms solutions comparison

Algorithm	Time, sec	Iterations, num	Opt Obj Function
Simplex	0.0035495758	5	0.9667
Ellipsoid	0.042	324	1.0019
Projective	0.0038997	17	0.9918

We have solved our linear programming problem with 3 different algorithms. The fastest algorithm was Simplex method, the second fastest was Projective algorithm. Moreover, for simplex method it takes only 5 iterations to find optimal solution to our problem. However, due to all 3 algorithms were implemented on 3 different computers with different configurations, time required for optimization may be different, in case we would use one computer for algorithm implementation.

Literature

1. https://www.math.ucla.edu/~tom/LP.pdf
2. http://repobib.ubiobio.cl/jspui/bitstream/123456789/282/3/Chavez_Abello_Rodrigo.pdf
3. https://math.mit.edu/~goemans/18310S15/lpnotes310.pdf
4. http://fourier.eng.hmc.edu/e176/lectures/NM/node32.html
5. https://personal.utdallas.edu/~metin/Or6201/simplex_s.pdf
6. Lagarias, J. C., and R. J. Vanderbei. "II Dikin's convergence result for the affine scaling algorithm." *Contemporary Math* 114 (1990): 109-119.
7. Karmarkar, Narendra. "A new polynomial-time algorithm for linear programming." *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. ACM, 1984.
8. Rebennack S. (2008) Ellipsoid Method. In: Floudas C., Pardalos P. (eds) Encyclopedia of Optimization. Springer, Boston
9. http://www-math.mit.edu/~goemans/18433S09/ellipsoid.pdf
10. https://www.cs.princeton.edu/courses/archive/fall05/cos521/ellipsoid.pdf
11. https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/notes/lecture08.pdf
12. https://www.coursera.org/lecture/advanced-algorithms-and-complexity/optional-the-ellipsoid-algorithm-N9rzA