CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

Decision Analytics

Lecture 18-19: Linear programming

Linear programming

- Constraint programming was looking at combinatorial problems, i.e. problems where the domain is discrete, i.e. $D \subset \mathbb{Z}^n$

- We will now go back to a class of optimisation problems where the domain of the variables is again continuous, i.e. $D \subset \mathbb{R}^n$

Linear programming

- A linear program seeks find the vector x that maximises or minimises a given linear objective function

$$f[x] = c_0 + c_1 x_1 + \cdots + c_n x_n$$

- Subject to linear equality and inequality constraints, such as

$$\begin{gathered} a_{i1} x_1 + \cdots + a_{in} x_n = b_i \\ a_{j1} x_1 + \cdots + a_{jn} x_n \leq b_j \\ a_{k1} x_1 + \cdots + a_{kn} x_n \geq b_k \end{gathered}$$

- Some (not necessarily all) variables might be restricted to be positive

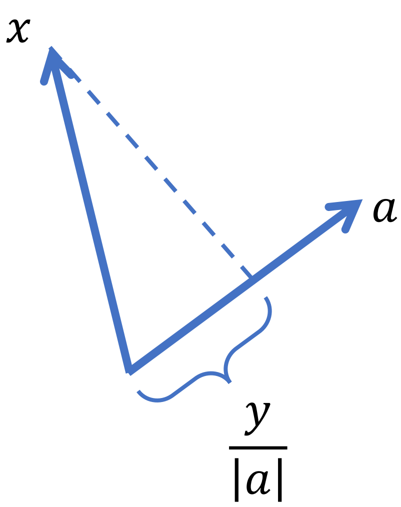
$$x_s \geq 0$$

Scalar products

- All these linear functions and constraints are scalar products between vectors
- So, let's try to get some geometric insight into a scalar product $y = a^T x$

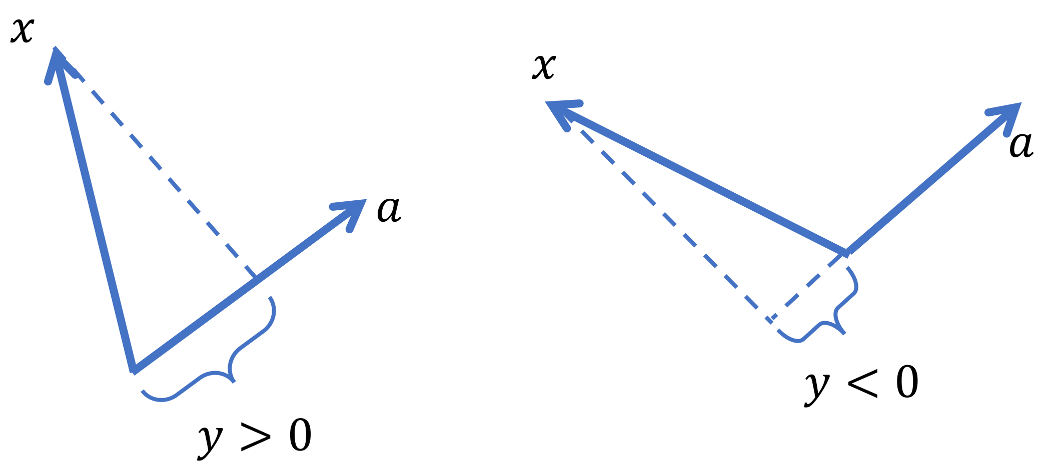
Scalar products

- All these linear functions and constraints are scalar products between vectors
- So, let's try to get some geometric insight into a scalar product $y = a^T x$
  - It is the length of the normalised orthogonal projection of x on a



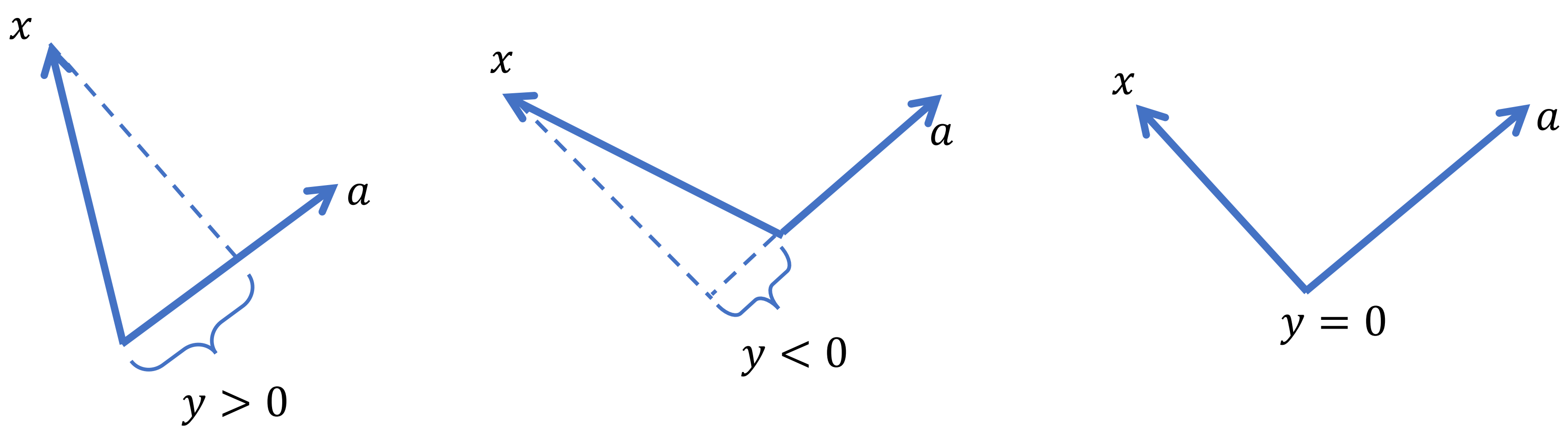
Scalar products

- All these linear functions and constraints are scalar products between vectors
- So, let's try to get some geometric insight into a scalar product $y = a^T x$
  - It is the length of the normalised orthogonal projection of x on a
  - In particular, it is positive if it is more towards the direction of a and negative otherwise



Scalar products

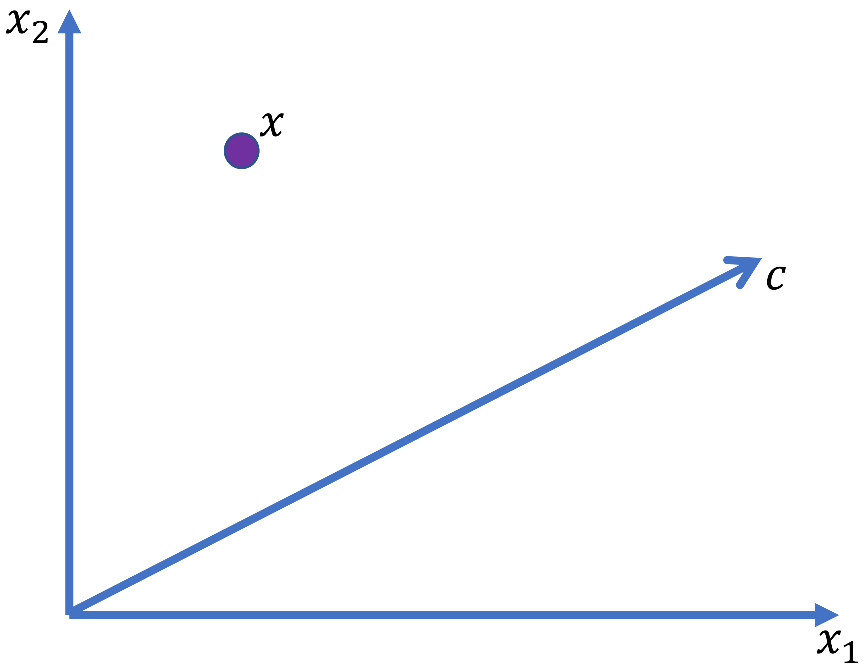
- All these linear functions and constraints are scalar products between vectors
- So, let's try to get some geometric insight into a scalar product $y = a^T x$
  - It is the length of the normalised orthogonal projection of x on a
  - In particular, it is positive if it is more towards the direction of a and negative otherwise
  - It is zero if both vectors are perpendicular



Geometric intuition

- Let's now look at the objective function first

$$f[x] = c^T x$$

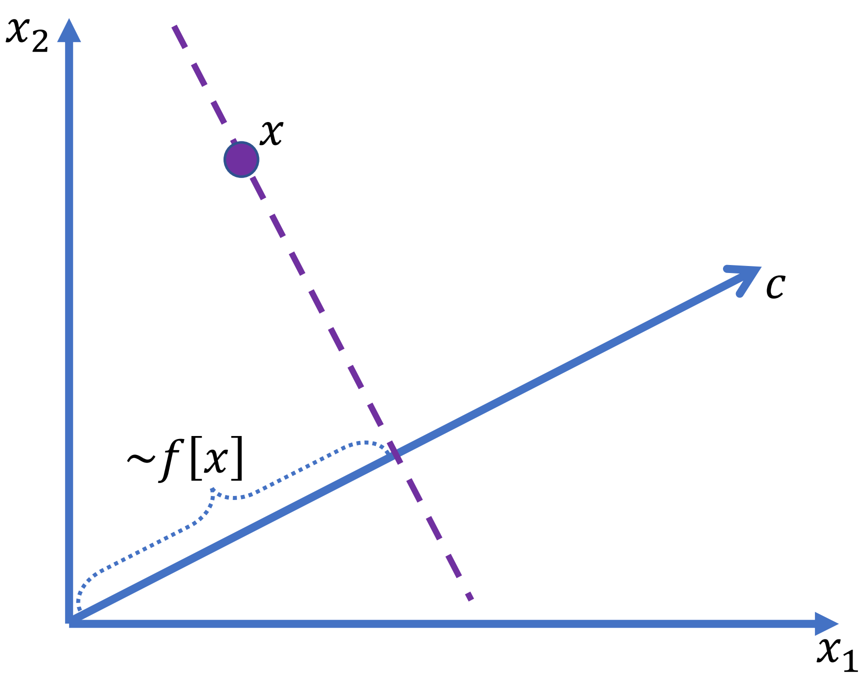


Geometric intuition

- Let's now look at the objective function first

$$f[x] = c^T x$$

- The value of the objective function $f[x]$ is proportional to the projection on c

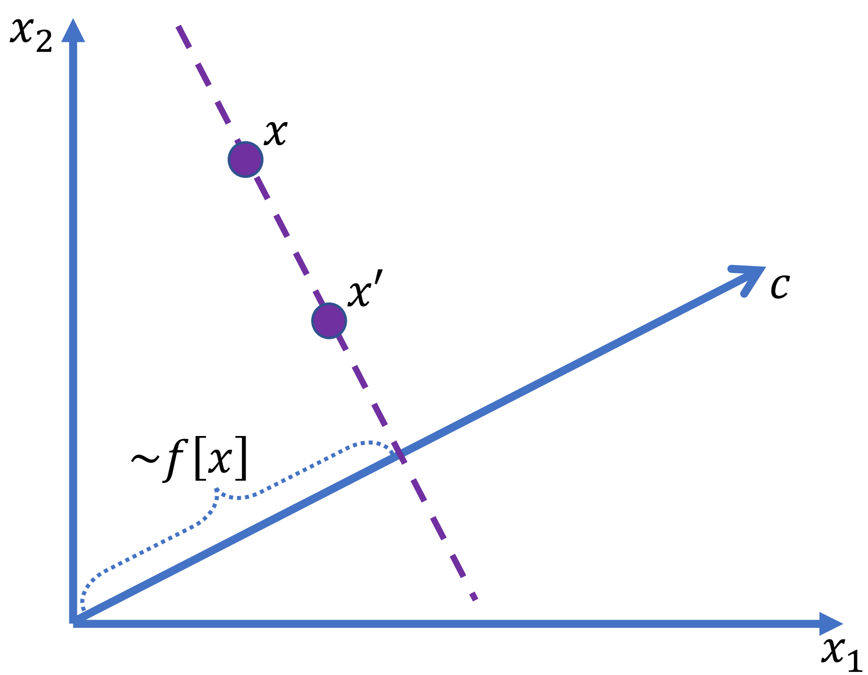


Geometric intuition

- Let's now look at the objective function first

$$f[x] = c^T x$$

- The value of the objective function $f[x]$ is proportional to the projection on c
- Also, $f[x] = f[x']$ for all x' on the hyperplane through x perpendicular to c

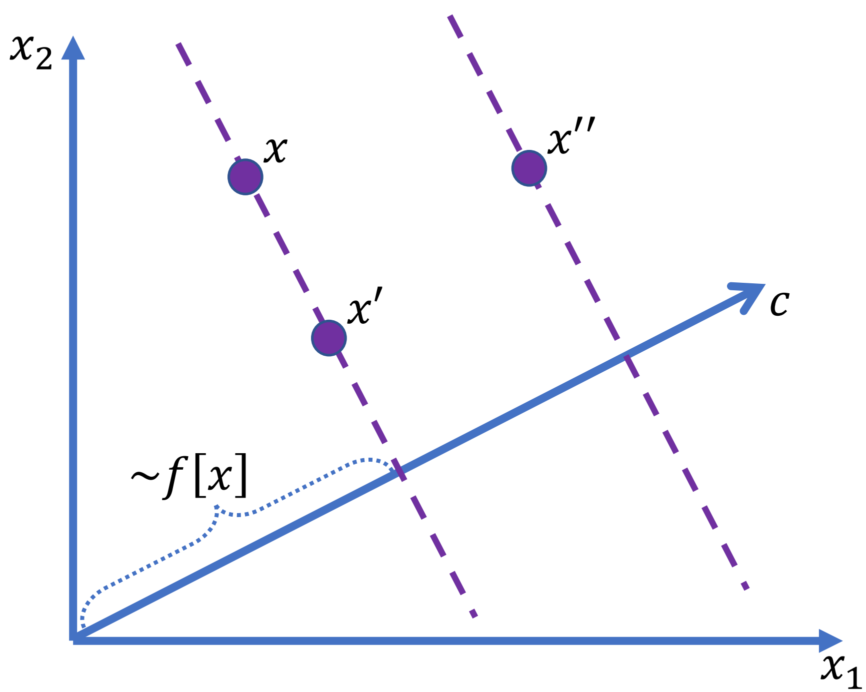


Geometric intuition

- Let's now look at the objective function first

$$f[x] = c^T x$$

- The value of the objective function $f[x]$ is proportional to the projection on c
- Also, $f[x] = f[x']$ for all x' on the hyperplane through x perpendicular to c
- Finally, $f[x''] > f[x]$ for all x'' on hyperplanes further along c

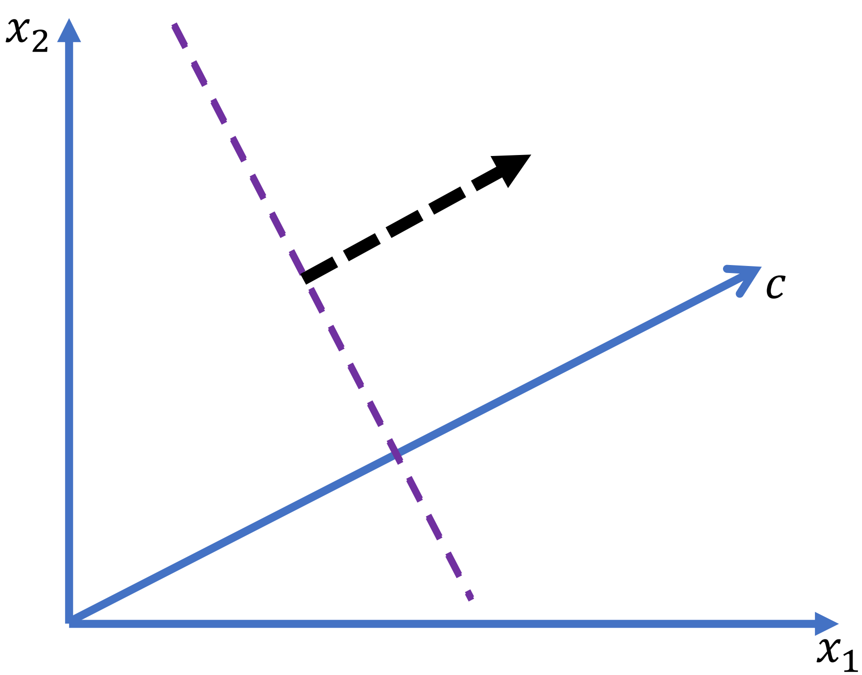


Geometric intuition

- Let's now look at the objective function first

$$f[x] = c^T x$$

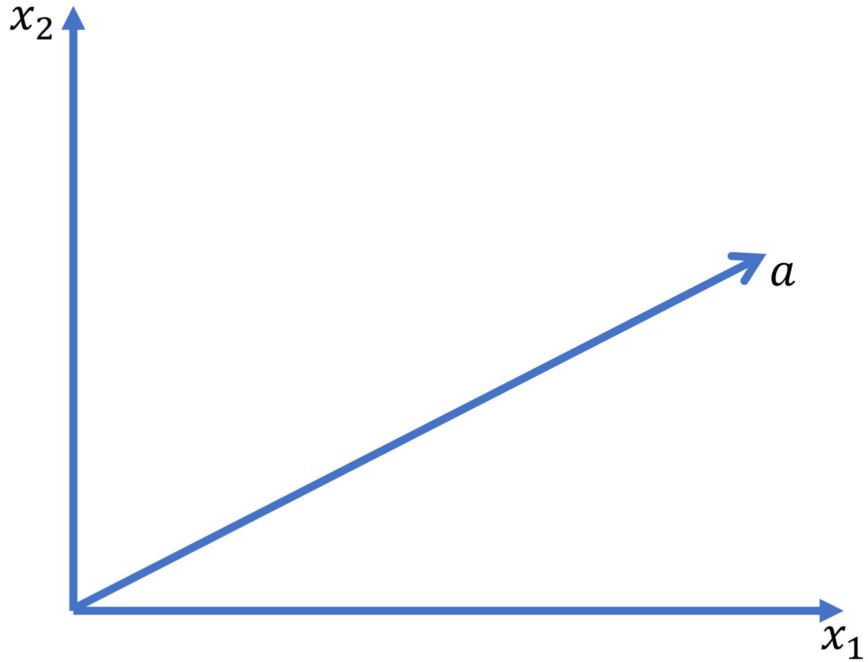
- The geometric intuition of maximising $f[x]$ is therefore to push a hyperplane perpendicular to c further out into the direction of c



Geometric intuition

- Let's continue with the linear equality constraints

$$ax \leq b$$

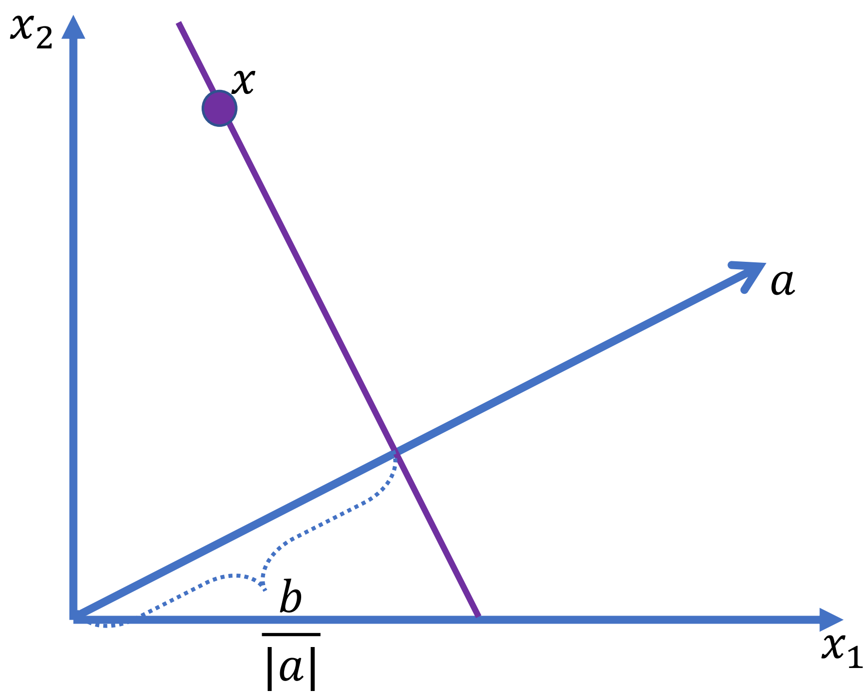


Geometric intuition

- Let's continue with the linear equality constraints

$$ax \leq b$$

- $ax = b$ for all points x on a hyperplane perpendicular to a

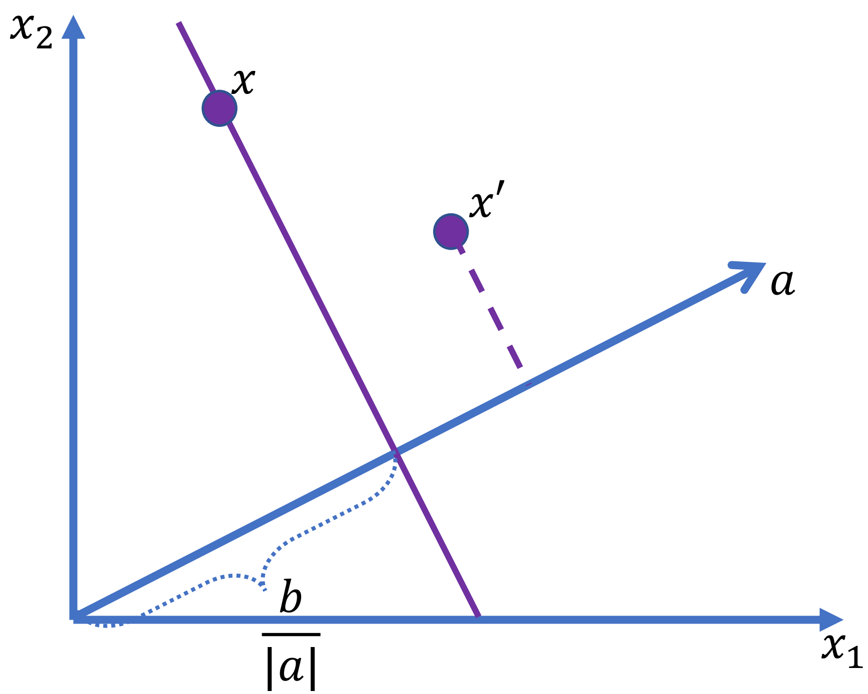


Geometric intuition

- Let's continue with the linear equality constraints

$$ax \leq b$$

- $ax = b$ for all points x on a hyperplane perpendicular to a
- $ax' \geq b$ for all points x' further out from the hyperplane in the direction of a

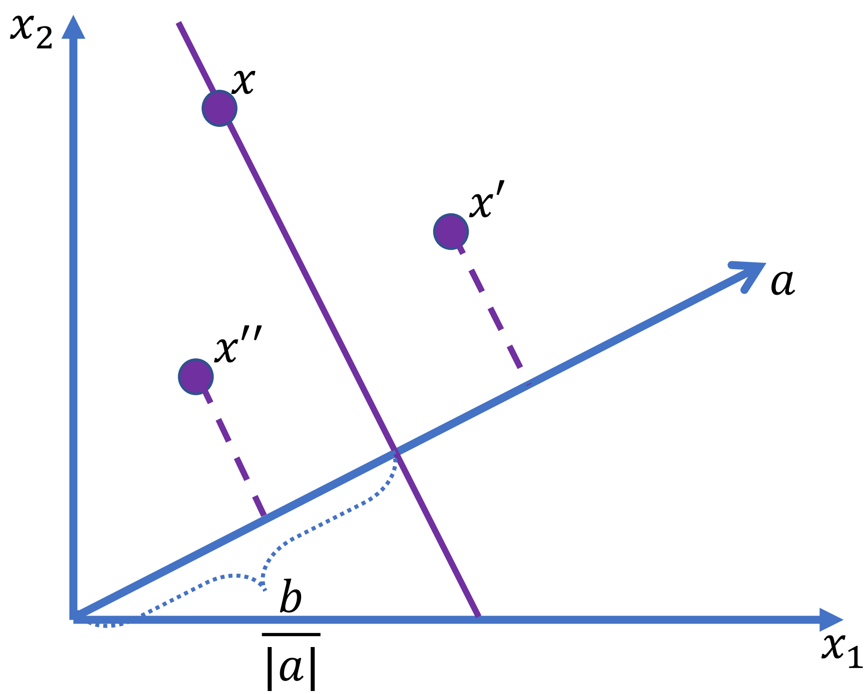


Geometric intuition

- Let's continue with the linear equality constraints

$$ax \leq b$$

- $ax = b$ for all points x on a hyperplane perpendicular to a
- $ax' \geq b$ for all points x' further out from the hyperplane in the direction of a
- $ax'' \leq b$ for all points x'' on the opposite side of the hyperplane

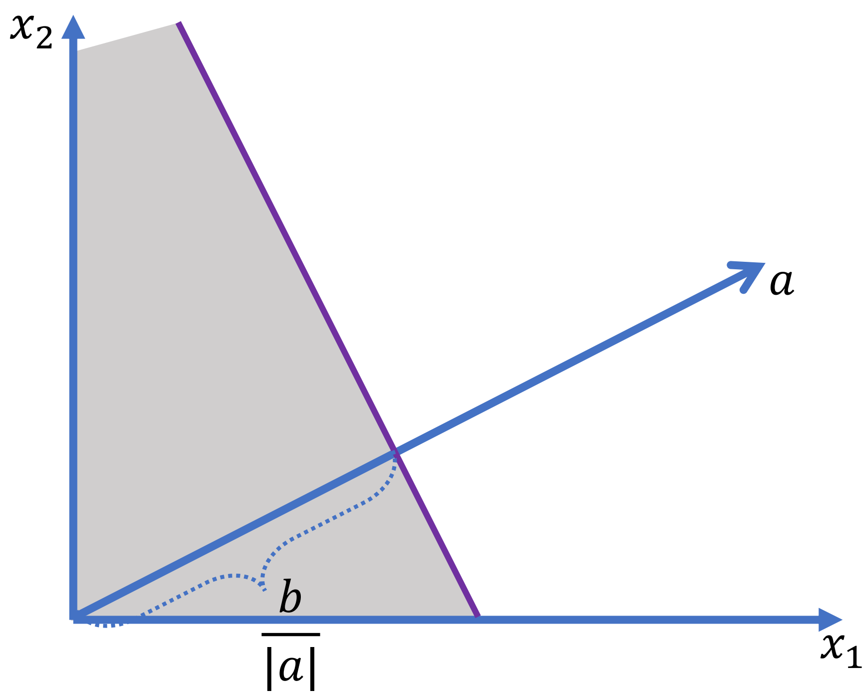


Geometric intuition

- Let's continue with the linear equality constraints

$$ax \leq b$$

- The geometric intuition of a linear inequality constraint is therefore to restrict feasible solutions to the area on one side of a hyperplane perpendicular to a at a distance of $\frac{b}{|a|}$ to the coordinate origin

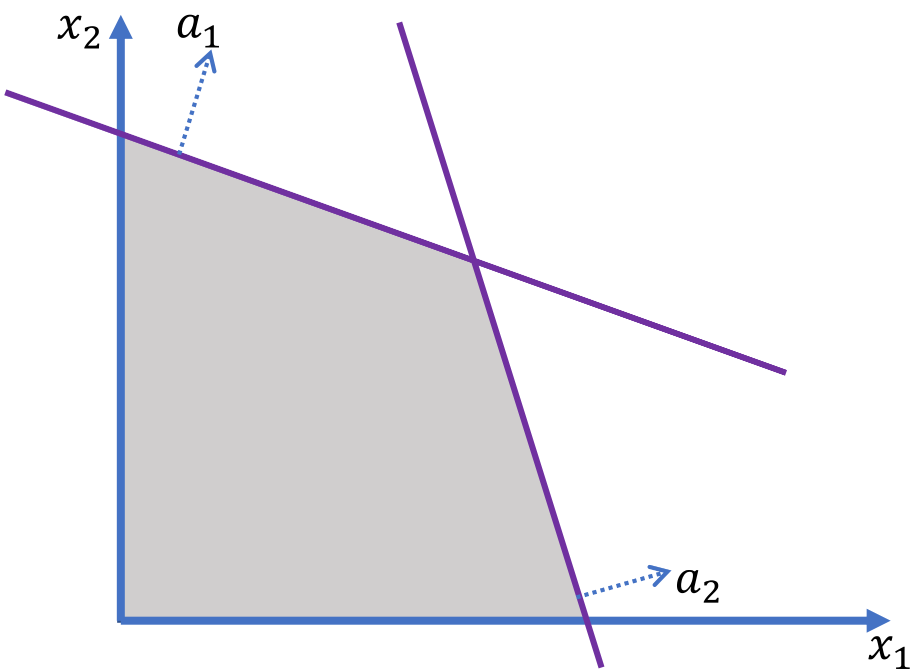


Geometric intuition

- In case there are more linear inequalities than dimensions, e.g.

$$a_1 x \leq b_1$$
$$a_2 x \leq b_2$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

- The feasible solution space becomes a simplex

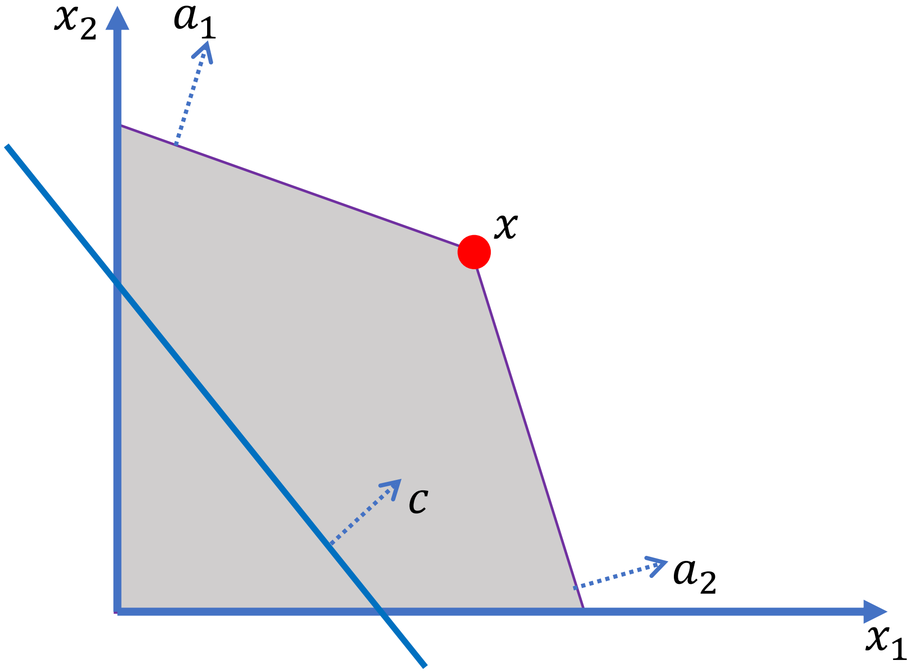


Geometric intuition

- In case there are more linear inequalities than dimensions, e.g.

$$a_1 x \leq b_1$$
$$a_2 x \leq b_2$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

- The feasible solution space becomes a simplex
- The optimal solution maximising $c^T x$ is the corner point on the tangential hyperplane perpendicular to c



Standard form

- The insight that the solution has to be on the border of the simplex can be formalised by stating that every linear program can be converted into standard form

- A linear program is stated in **standard form** when it is maximising a linear objective function

$$f = c_0 + c^T x$$

- Subject to the linear constraints

$$\begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned}$$

- We will see now how we can transform every general linear programming problem into a linear programming problem in standard form (some solvers require standard form)

Standard form

- First, if the problem is to minimise (instead of maximise)

$$f[x] = c_0 + c^T x$$

- we can transform it into a maximisation problem by substituting

$$c' = -c$$

- and then maximising

$$f'[x] = -c_0 + c'^T x$$

Standard form

- For every inequality of the form

$$a^T x \leq b$$

- we can add an additional **slack variable** s and use the following two constraints instead

$$a^T x + s = b$$

$$s \geq 0$$

Standard form

- Similarly, for every inequality of the form

$$a^T x \geq b$$

- we can add an additional **surplus variable** s and use the following two constraints instead

$$a^T x - s = b$$

$$s \geq 0$$

Standard form

- Finally, if a component x_i is to be unbound we can introduce two new **decision variables** x_i^+ and x_i^- instead, which are to be positive

$$x_i^+ \geq 0$$
$$x_i^- \geq 0$$

- All we need to do then is replace all occurrences of x_i with

$$x_i = x_i^+ - x_i^-$$

The diet problem

- A common LP task is called the diet problem
- For example, in farming it is important to determine the most cost effective feed mix that is meeting the targeted nutritional requirements
- If for example we can choose between two feeds

Feed	Energy	Protein	Calcium	Cost
A	2	5	4	9
B	4	3	1	7

- We need to achieve at least the following nutritional composition to make sure the final product achieves the targeted quality standard

Energy	Protein	Calcium
12	15	8

- What is the cost-optimal feed mix under these constraints?

The diet problem

- The feed mix can be modelled using x_1 units of A and x_2 units of B
- We then need to minimise the cost

$$f = 9x_1 + 7x_2$$

- subject to the constraints (the last two because we cannot feed negative amounts)

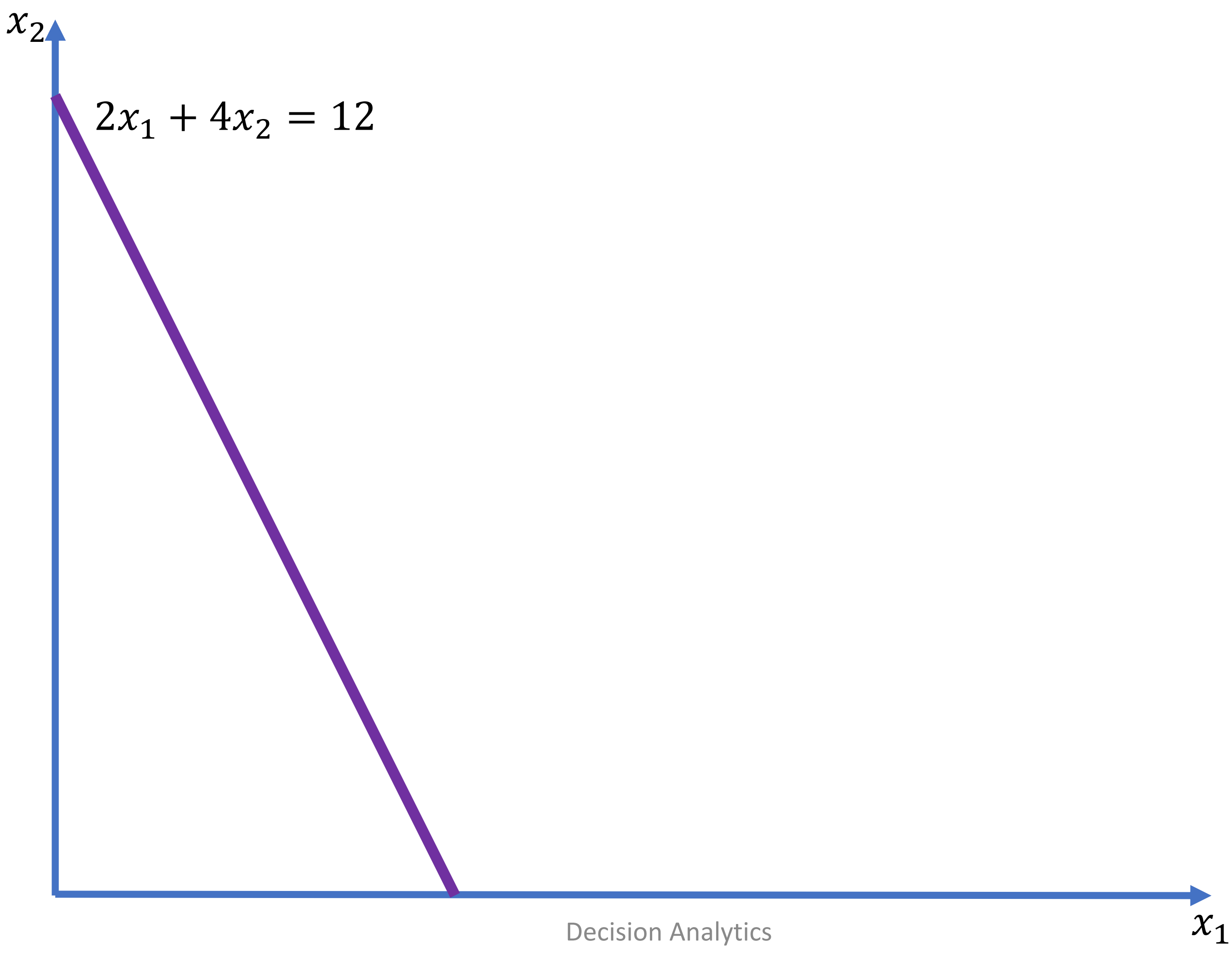
$$2x_1 + 4x_2 \geq 12$$
$$5x_1 + 3x_2 \geq 15$$
$$4x_1 + x_2 \geq 8$$

$$x_1 \geq 0, x_2 \geq 0$$

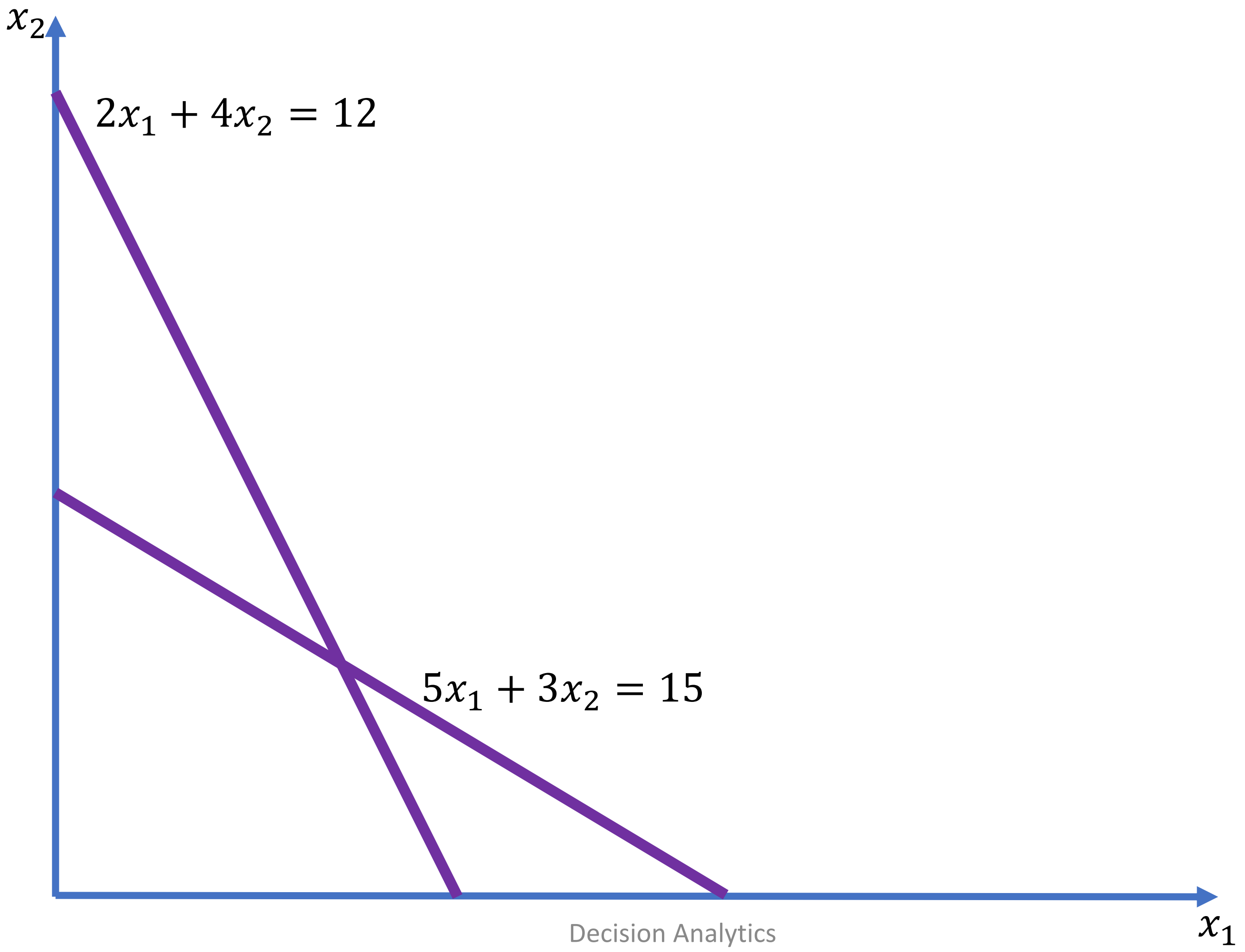
Feed	Energy	Protein	Calcium	Cost
A	2	5	4	9
B	4	3	1	7

Energy	Protein	Calcium
12	15	8

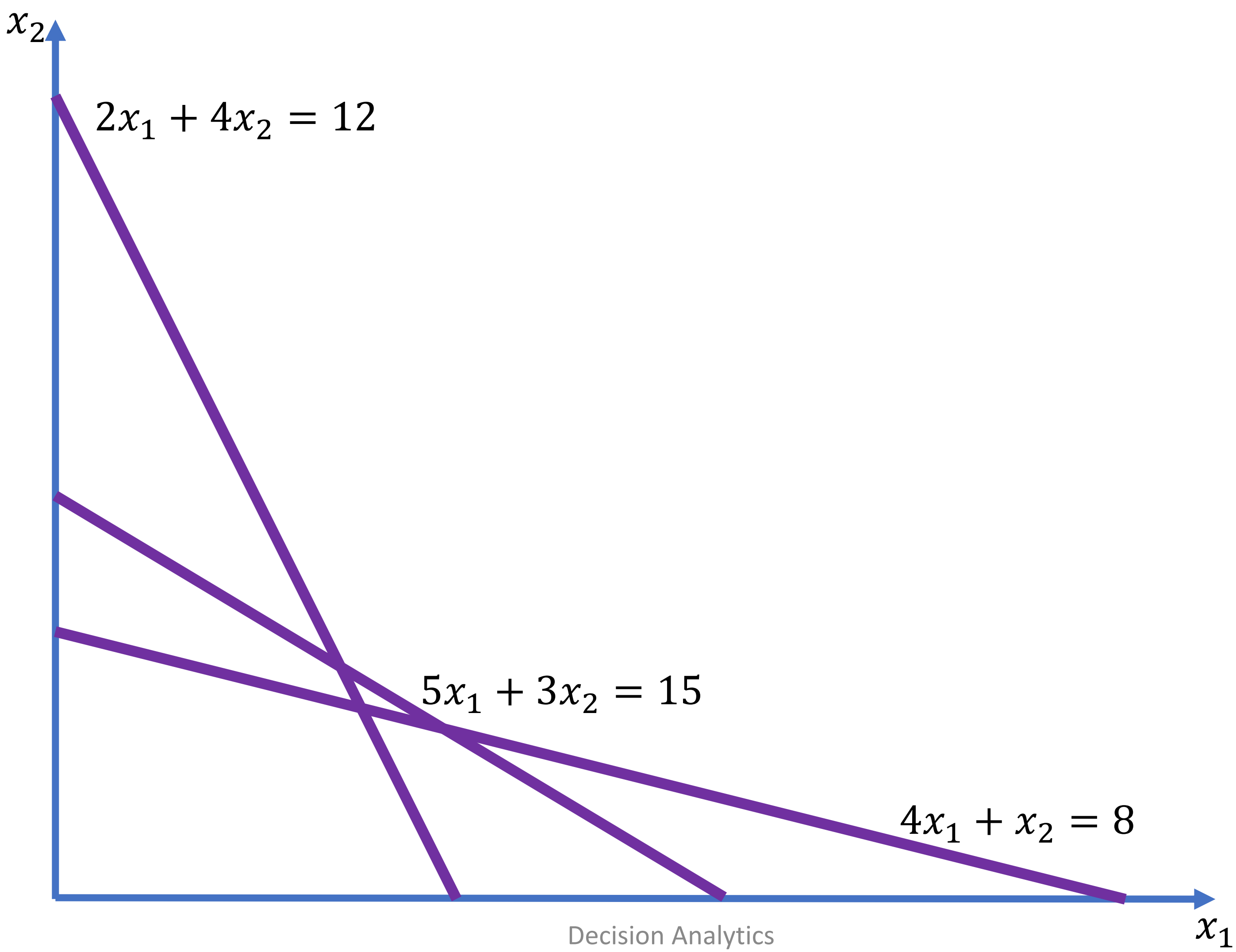
The diet problem



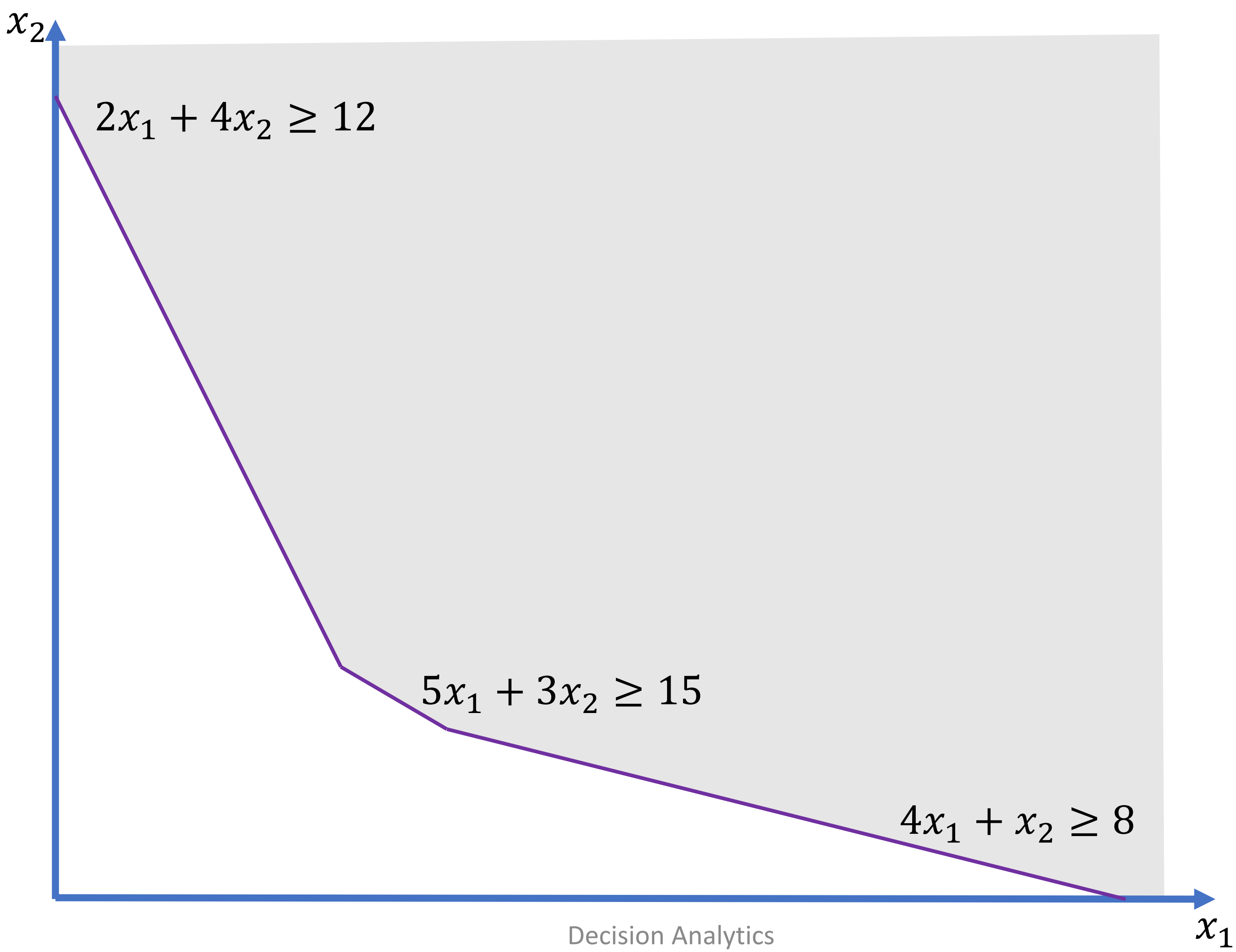
The diet problem



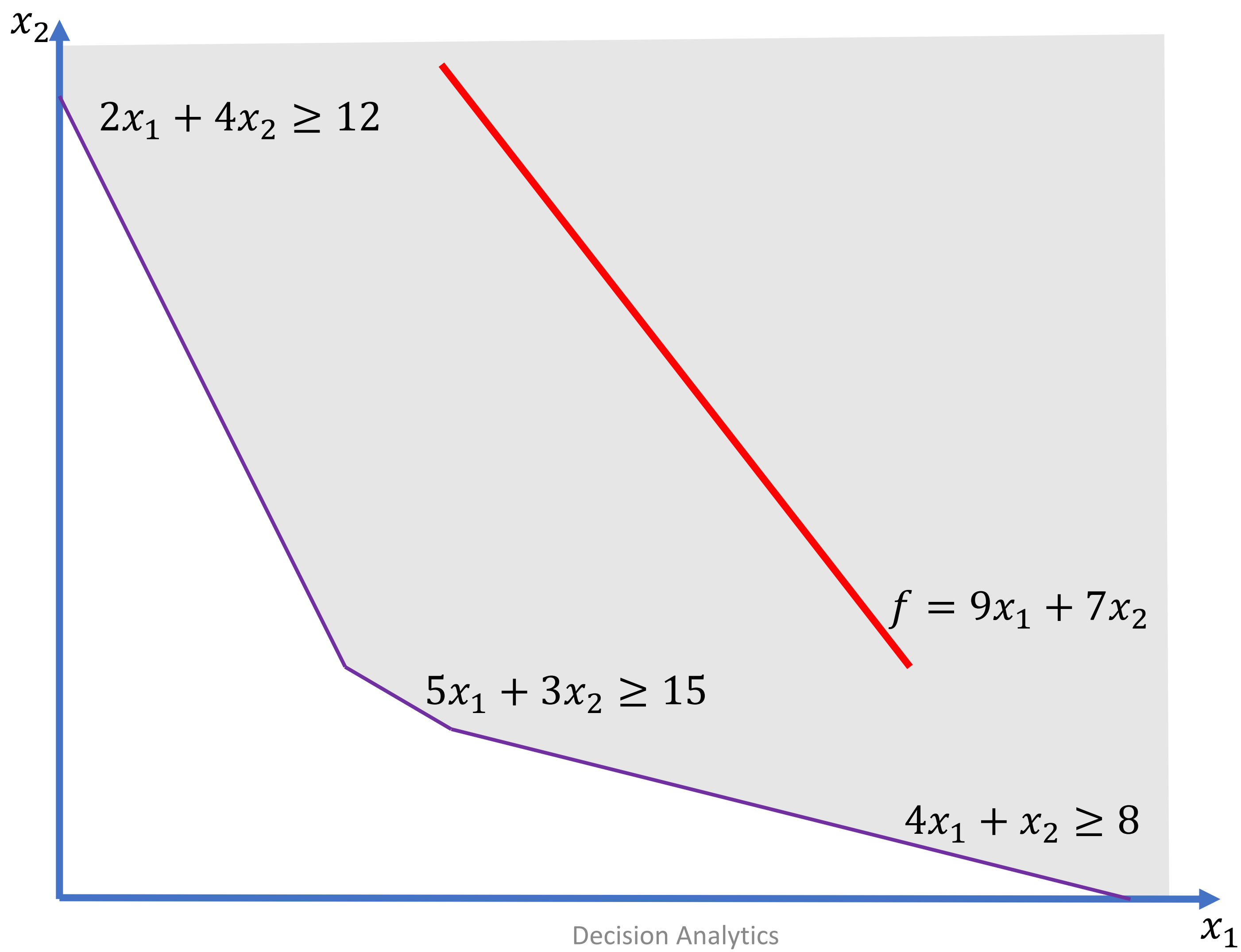
The diet problem



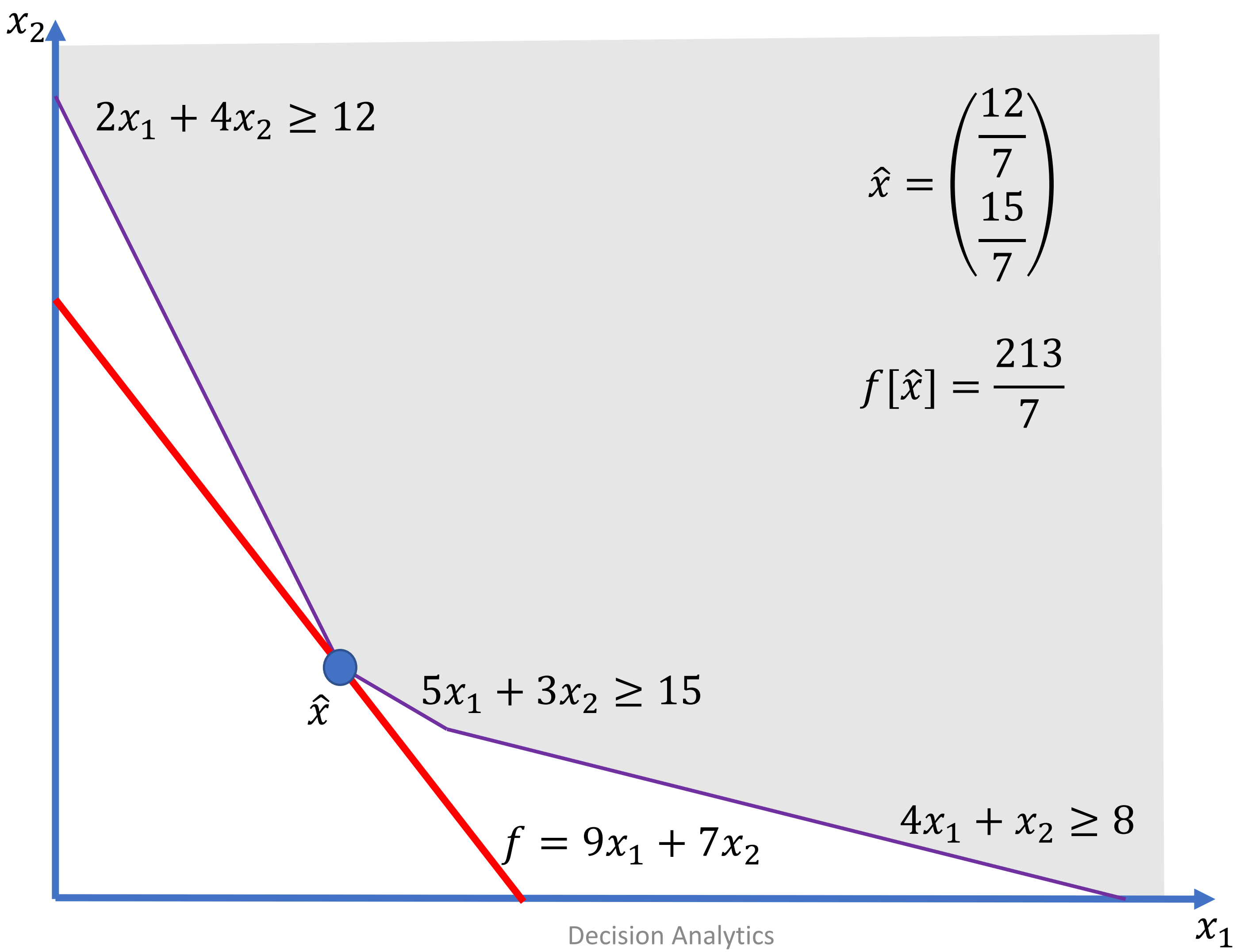
The diet problem



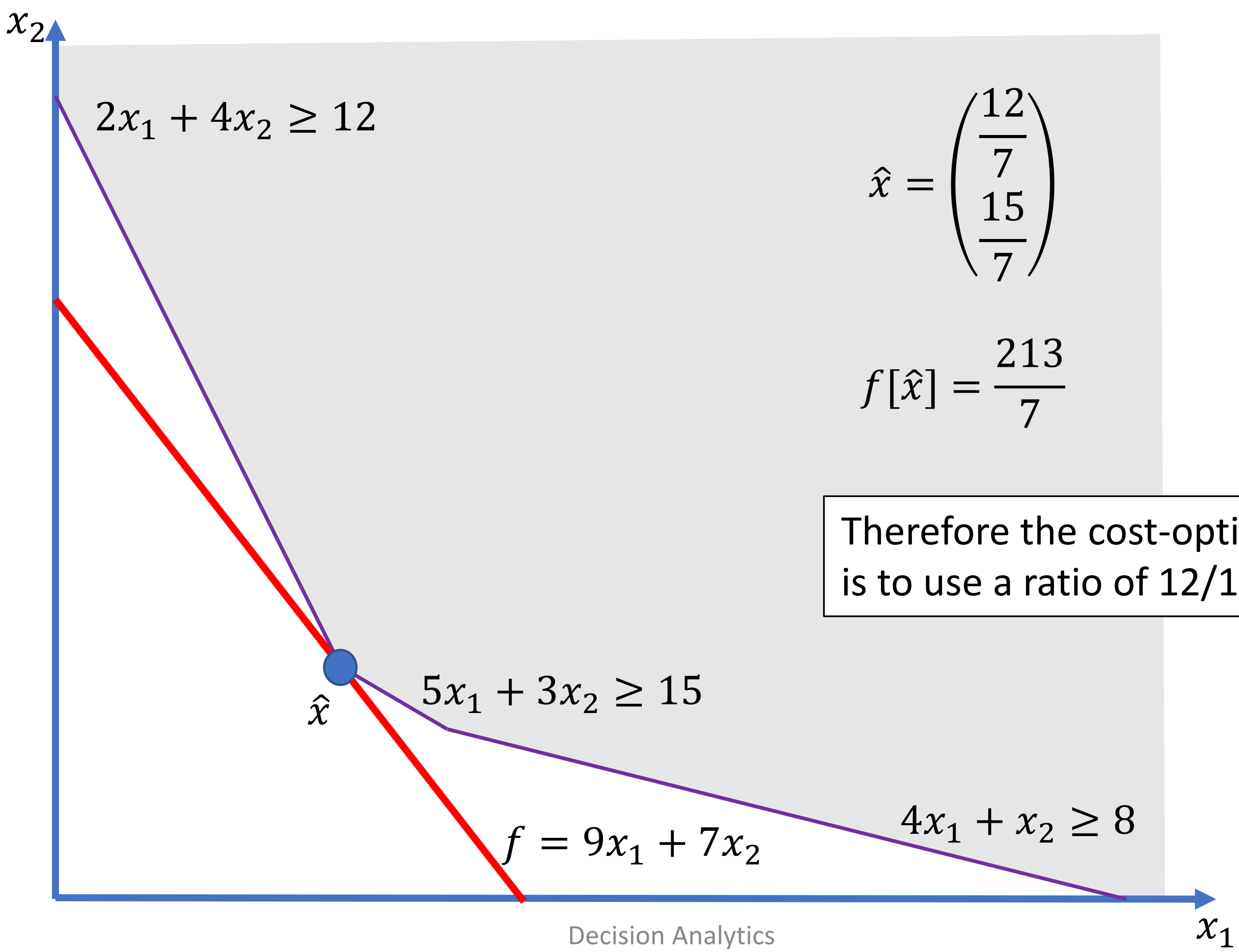
The diet problem



The diet problem



The diet problem



Therefore the cost-optimal feed mix is to use a ratio of 12/15 in this case.

The diet problem

- To solve this problem using OR Tools we can use the GLOP wrapper

```
from ortools.linear_solver import pywraplp

solver = pywraplp.Solver('LPWrapper',
            pywraplp.Solver.GLOP_LINEAR_PROGRAMMING)
```

The diet problem

- To solve this problem using OR Tools we can use the GLOP wrapper
- We define the two model variables $0 \leq x_1 \leq \infty$ and $0 \leq x_2 \leq \infty$

```
x1 = solver.NumVar(0, solver.infinity(), 'x1')
x2 = solver.NumVar(0, solver.infinity(), 'x2')
```

The diet problem

- To solve this problem using OR Tools we can use the GLOP wrapper
- We define the two model variables x_1 and x_2 with their lower and upper bound
- We add the first constraint $12 \leq 2x_1 + 4x_2 \leq \infty$

```
c1 = solver.Constraint(12, solver.infinity())
c1.SetCoefficient(x1, 2)
c1.SetCoefficient(x2, 4)
```

The diet problem

- To solve this problem using OR Tools we can use the GLOP wrapper
- We define the two model variables x_1 and x_2 with their lower and upper bound
- We add the first constraint $12 \leq 2x_1 + 4x_2 \leq \infty$
- Then we add the second constraint $15 \leq 5x_1 + 3x_2 \leq \infty$

```
c2 = solver.Constraint(15, solver.infinity())
c2.SetCoefficient(x1, 5)
c2.SetCoefficient(x2, 3)
```

The diet problem

- To solve this problem using OR Tools we can use the GLOP wrapper
- We define the two model variables x_1 and x_2 with their lower and upper bound
- We add the first constraint $12 \leq 2x_1 + 4x_2 \leq \infty$
- Then we add the second constraint $15 \leq 5x_1 + 3x_2 \leq \infty$
- And the third constraint $8 \leq 4x_1 + 1x_2 \leq \infty$

```
c3 = solver.Constraint(8, solver.infinity())
c3.SetCoefficient(x1, 4)
c3.SetCoefficient(x2, 1)
```

The diet problem

- To solve this problem using OR Tools we can use the GLOP wrapper
- We define the two model variables x_1 and x_2 with their lower and upper bound
- We add the first constraint $12 \leq 2x_1 + 4x_2 \leq \infty$
- Then we add the second constraint $15 \leq 5x_1 + 3x_2 \leq \infty$
- And the third constraint $8 \leq 4x_1 + 1x_2 \leq \infty$
- Finally, we add the objective function $9x_1 + 7x_2$ and solve as minimisation problem

```
objective = solver.Objective()
objective.SetCoefficient(x1, 9)
objective.SetCoefficient(x2, 7)
objective.SetMinimization()
solver.Solve()
```

The diet problem

- To solve this problem using OR Tools we can use the GLOP wrapper
- We define the two model variables x_1 and x_2 with their lower and upper bound
- We add the first constraint $12 \leq 2x_1 + 4x_2 \leq \infty$
- Then we add the second constraint $15 \leq 5x_1 + 3x_2 \leq \infty$
- And the third constraint $8 \leq 4x_1 + 1x_2 \leq \infty$
- Finally, we add the objective function $9x_1 + 7x_2$ and solve as minimisation problem
- Extracting the result from the solver we get

$$x_1 = 1.7, \qquad x_2 = 2.1, \qquad f = 30.4$$

```
print ("x1 = " , x1.solution_value())
print ("x2 = ",  x2.solution_value())
print("cost = ", (9 * x1.solution_value() + 7 * x2.solution_value()))
```

The transportation problem

- A second common LP problem is matching supply and demand in a scenario where transport costs between supplier and consumer varies

- For example let there be two suppliers of electricity and three consumers
- The suppliers produce 6MW and 9MW respectively
- The consumers demand is 8MW, 5MW, and 2MW (if there is no storage, supply and demand have to match up)
- Transportation costs between supplier and consumer via the grid are different

	Consumer A	Consumer B	Consumer C
Supplier A	5	5	3
Supplier B	6	4	1

- What is the cost-optimal energy mix for each consumer?

The transportation problem

- The decision variables in this case are the quantities transported between each supplier and consumer, i.e. $x_{ij}, i = \{1,2\}, j = \{1,2,3\}$
- The total transportation cost, and hence the objective function, is
$$f[x] = 5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23}$$
- First, the demand must be met and therefore
$$x_{11} + x_{21} = 8$$
$$x_{12} + x_{23} = 5$$
$$x_{13} + x_{23} = 2$$
- Then, the suppliers can not produce over capacity
$$x_{11} + x_{12} + x_{13} = 6$$
$$x_{21} + x_{22} + x_{23} = 9$$
- And finally, all transports must be positive
$$x_{11} \geq 0, x_{12} \geq 0, x_{13} \geq\ 0$$
$$x_{21} \geq 0, x_{22} \geq 0, x_{23} \geq\ 0$$

	Supply
Supplier A	6
Supplier B	9

	Demand
Consumer A	8
Consumer B	5
Consumer C	2

	Consumer A	Consumer B	Consumer C
Supplier A	5	5	3
Supplier B	6	4	1

The transportation problem

- We use the OR Tools GLOP wrapper again
- We define the six model variables $0 \leq x_{ij} \leq \infty$

```
solver = pywraplp.Solver('LPWrapper',
              pywraplp.Solver.GLOP_LINEAR_PROGRAMMING)

x11 = solver.NumVar(0, solver.infinity(), 'x11')
x12 = solver.NumVar(0, solver.infinity(), 'x12')
x13 = solver.NumVar(0, solver.infinity(), 'x13')
x21 = solver.NumVar(0, solver.infinity(), 'x21')
x22 = solver.NumVar(0, solver.infinity(), 'x22')
x23 = solver.NumVar(0, solver.infinity(), 'x23')
```

The transportation problem

- We use the OR Tools GLOP wrapper again
- We define the six model variables $0 \leq x_{ij} \leq \infty$
- Then we add the constraints $x_{11} + x_{21} = 8$, $x_{12} + x_{22} = 5$ and $\mathrm{x}_{13} + x_{23} = 2$

```
c1 = solver.Constraint(8, 8)
c1.SetCoefficient(x11, 1)
c1.SetCoefficient(x21, 1)

c1 = solver.Constraint(5, 5)
c1.SetCoefficient(x12, 1)
c1.SetCoefficient(x22, 1)

c3 = solver.Constraint(2, 2)
c3.SetCoefficient(x13, 1)
c3.SetCoefficient(x23, 1)
```

The transportation problem

- We use the OR Tools GLOP wrapper again
- We define the six model variables $0 \leq x_{ij} \leq \infty$
- Then we add the constraints $x_{11} + x_{21} = 8$, $x_{12} + x_{22} = 5$ and $x_{13} + x_{23} = 2$
- The same for $x_{11} + x_{12} + x_{13} = 6$ and $x_{21} + x_{22} + x_{23} = 9$

```
c4 = solver.Constraint(6, 6)
c4.SetCoefficient(x11, 1)
c4.SetCoefficient(x12, 1)
c4.SetCoefficient(x13, 1)

c5 = solver.Constraint(9, 9)
c5.SetCoefficient(x21, 1)
c5.SetCoefficient(x22, 1)
c5.SetCoefficient(x23, 1)
```

The transportation problem

- We use the OR Tools GLOP wrapper again
- We define the six model variables $0 \leq x_{ij} \leq \infty$
- Then we add the constraints $x_{11} + x_{21} = 8$, $x_{12} + x_{22} = 5$ and $x_{13} + x_{23} = 2$
- The same for $x_{11} + x_{12} + x_{13} = 6$ and $x_{21} + x_{22} + x_{23} = 9$
- Finally the objective function $5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23}$ and solve

```
objective = solver.Objective()
objective.SetCoefficient(x11, 5)
objective.SetCoefficient(x12, 5)
objective.SetCoefficient(x13, 3)
objective.SetCoefficient(x21, 6)
objective.SetCoefficient(x22, 4)
objective.SetCoefficient(x23, 1)
objective.SetMinimization()
solver.Solve()
```

The transportation problem

- We use the OR Tools GLOP wrapper again
- We define the six model variables $0 \leq x_{ij} \leq \infty$
- Then we add the constraints $x_{11} + x_{21} = 8$, $x_{12} + x_{22} = 5$ and $x_{13} + x_{23} = 2$
- The same for $x_{11} + x_{12} + x_{13} = 6$ and $x_{21} + x_{22} + x_{23} = 9$
- Finally the objective function $5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23}$ and solve
- The cost-optimal transportation through the grid is then

```
print ("S1C1 = " , x11.solution_value())
print ("S1C2 = " , x12.solution_value())
print ("S1C3 = " , x13.solution_value())
print ("S2C1 = " , x21.solution_value())
print ("S2C2 = " , x22.solution_value())
print ("S2C3 = " , x23.solution_value())
```

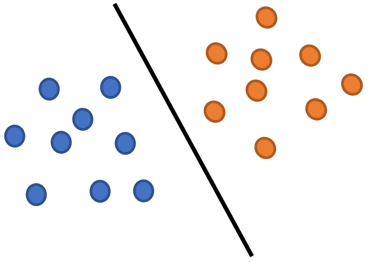
```
S1C1 =  6.0
S1C2 =  0.0
S1C3 =  0.0
S2C1 =  2.0
S2C2 =  5.0
S2C3 =  2.0
cost =  64.0
```

Linear discriminant function

- In machine learning the perceptron function is used to discriminate between two classes of objects by separating them in feature space with a hyperplane

- The class of an object ω is determined by testing the sign of the scalar product between the feature vector x and a weight vector w

$$\omega = \begin{cases} 1 & w^T x \geq 0 \\ -1 & w^T x < 0 \end{cases}$$



- Learning the weight vector w (geometrically this means the separating hyperplane) from training samples $\{(x_1, \omega_1), \dots, (x_n, \omega_n)\}$ enables to decide for new points x which class ω they belong to

- The perceptron learning problem can be formulated as a linear program

Linear discriminant function

- With the substitution $y_i = \omega_i x_i$ a sample from the training set is classified incorrectly, if

$$w^T y_i < 0$$

- The Perceptron criterion function now sums up the “distances” from the hyperplane for all these misclassified sample points

$$J[w] = \sum_{y_i \in \{\omega_i x_i | \omega_i x_i < 0\}} -w^T y_i$$

- To avoid the trivial solution $w = 0$ we need to introduce a margin ϵ to the hyperplane and instead minimise

$$J'[w] = \sum_{y_i \in \{\omega_i x_i | \omega_i x_i < \epsilon\}} \epsilon - w^T y_i$$

Linear discriminant function

- A minimum of

$$J'[w] = \sum_{y_i \in \{\omega_i x_i | \omega_i x_i < \epsilon\}} \epsilon - w^T y_i$$

- can be stated as LP by observing that if we introduce additional variables τ_i and minimise the linear function in both the weights and these variables

$$f[w, \tau] = 0^T w + \sum_{i=1}^{n} \tau_i$$

- Subject to the linear inequalities

$$\tau_i \geq \epsilon - w^T y_i$$
$$\tau_i \geq 0$$

- We minimise every summand in the above cost function and achieve an optimal weight vector (this also takes into account the correctly classified, but they do not matter in the perceptron criterion)

Linear discriminant function

- We use the OR Tools GLOP wrapper again
- We define the model variables $0 \leq \tau_i \leq \infty$ and $-\infty \leq w_i \leq \infty$

```
solver = pywraplp.Solver('LPWrapper',
                         pywraplp.Solver.GLOP_LINEAR_PROGRAMMING)

tau = [None]*n
for i in range(n):
    tau[i] = solver.NumVar(0, solver.infinity(), 'tau_'+str(i))

w = [None]*d
for j in range(d):
    w[j] = solver.NumVar(-solver.infinity(), solver.infinity(), 'w_'+str(j))
```

Linear discriminant function

- We use the OR Tools GLOP wrapper again
- We define the model variables $0 \leq \tau_i \leq \infty$ and $-\infty \leq w_i \leq \infty$
- We add the constraints $\tau_i \geq \epsilon - w^T y_i$

```
epsilon = 1e-6
for i in range(n):
    c = solver.Constraint(epsilon, solver.infinity())
    c.SetCoefficient(tau[i], 1)
    for j in range(d):
        c.SetCoefficient(w[j], y[i,j])
```

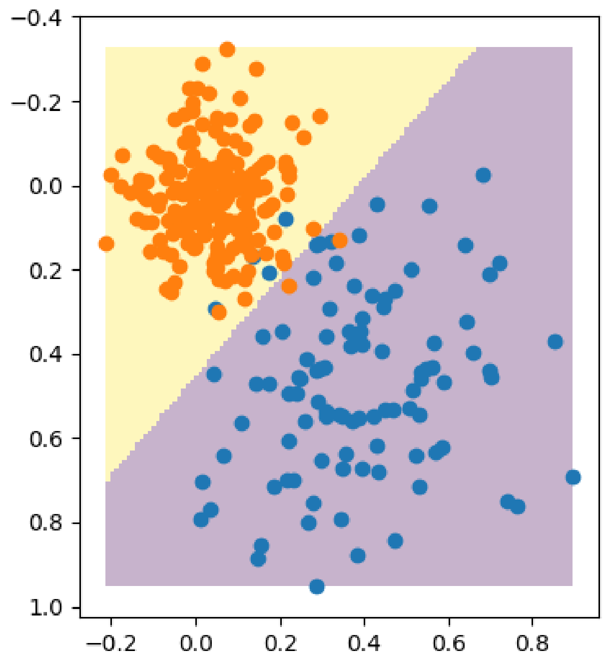
Linear discriminant function

- We use the OR Tools GLOP wrapper again
- We define the model variables $0 \leq \tau_i \leq \infty$ and $-\infty \leq w_i \leq \infty$
- We add the constraints $\tau_i \geq \epsilon - w^T y_i$
- Finally, we add the objective function $f[w, \tau] = 0^T w + \sum_{i=1}^{n} \tau_i$ and solve

```
f = solver.Objective()
for i in range(n):
    f.SetCoefficient(tau[i], 1)
f.SetMinimization()
solver.Solve()
```

Linear discriminant function

- We use the OR Tools GLOP wrapper again
- We define the model variables $0 \leq \tau_i \leq \infty$ and $-\infty \leq w_i \leq \infty$
- We add the constraints $\tau_i \geq \epsilon - w^T y_i$
- Finally, we add the objective function $f[w, \tau] = 0^T w + \sum_{i=1}^{n} \tau_i$ and solve
- The resulting weights define a linear boundary between two classes



Thank you for your attention!