CS 524

HW2 Sparsh Agarwal 9075905142

Q1. i) All inequalities will be <=

In [70]:
```
# 6 directions
d1 = [0; 0; 1]
d2 = [0; 0; -1]
d3 = [1; 0; 0]
d4 = [-1; 0; 0]
d5 = [0; 1; 0]
d6 = [0; -1; 0]

# faces
# face 1 : x=1 plane
v1 = d5 - d6
v2 = d2 - d1
f1 = cross(v1, v2)
# face 2 : x=-1 plane
v1 = d5 - d6
v2 = d1 - d2
f2 = cross(v1, v2)
# face 3 : y=1 plane
v1 = d1 - d2
v2 = d4 - d3
f3 = cross(v1, v2)
# face 4 : y=-1 plane
v1 = d1 - d2
v2 = d3 - d4
f4 = cross(v1, v2)
# face 5 : z=1 plane
v1 = d5 - d6
v2 = d3 - d4
f5 = cross(v1, v2)
# face 6 : z=-1 plane
v1 = d5 - d6
v2 = d4 - d3
f6 = cross(v1, v2)

b = zeros(6)
index = 1
for (i,j) in zip((f1, f2, f3, f4, f5, f6), (d3, d4, d5, d6, d1, d2))
        b[index] = i' * j
        index += 1
end
A = [f1[:];f2[:];f3[:];f4[:];f5[:];f6[:]]
println(A)
b
```

```
[-4, 0, 0, 4, 0, 0, 0, -4, 0, 0, 4, 0, 0, 0, -4, 0, 0, 4]
```

Out[70]:
```
6-element Array{Float64,1}:
 -4.0
 -4.0
 -4.0
 -4.0
 -4.0
 -4.0
```

ii) All inequalities will be <=, and distance of each plane from origin will be 1/1.732.

In [71]:
```
# 6 vertices
v1 = [0; 0; 1]
v2 = [0; 0; -1]
v3 = [1; 0; 0]
v4 = [-1; 0; 0]
v5 = [0; 1; 0]
v6 = [0; -1; 0]

# faces
# face 1
vec1 = v3 - v5
vec2 = v1 - v5
f1 = cross(vec1, vec2)
# face 2 :
vec1 = v1 - v5
vec2 = v4 - v5
f2 = cross(vec1, vec2)
# face 3 :
vec1 = v4 - v5
vec2 = v2 - v5
f3 = cross(vec1, vec2)
# face 4 :
vec1 = v2 - v5
vec2 = v3 - v5
f4 = cross(vec1, vec2)
# face 5 :
vec1 = v3 - v6
vec2 = v1 - v6
f5 = -cross(vec1, vec2)
# face 6 :
vec1 = v1 - v6
vec2 = v4 - v6
f6 = -cross(vec1, vec2)
# face 7 :
vec1 = v4 - v6
vec2 = v2 - v6
f7 = -cross(vec1, vec2)
# face 8 :
vec1 = v2 - v6
vec2 = v3 - v6
f8 = -cross(vec1, vec2)

b = zeros(6)
index = 1
for (i,j) in zip((f1, f2, f3, f4, f5, f6, f7, f8), (d3, d4, d5, d6, d1,
d2))
        b[index] = i' * j
        index += 1
end
A = [f1[:];f2[:];f3[:];f4[:];f5[:];f6[:]]
println(A)
b = [1/1.732,1/1.732,1/1.732,1/1.732,1/1.732,1/1.732,1/1.732]
println(b)
```

```
[-1, -1, -1, 1, -1, -1, 1, -1, 1, -1, -1, 1, -1, 1, -1, 1, 1, -1]
[0.577367, 0.577367, 0.577367, 0.577367, 0.577367, 0.577367, 0.577367]
```

Q2. A matrix would be = [0,0,1,0,0,1,0,0,0,0,0; 0,0,0,1,0,0,1,0,0,0,0; 0,0,0,0,1,0,0,1,0,0,0; 1,-1,-6,1,-1,0,0,0,1,0,0; 0,0,7,0,1,0,0,0,0,0,0; 0,0,0,1,1,0,0,0,0,1,0] b vector would be = [6,6,4,-4,14,5]
x vector would be = [u,v,w,x,y,s1,s2,s3,s4,s5,1] c vector would be = [-3,3,1,0,0,0,0,0,0,0,-1]

u-v -> z1, w-1 -> z2, x-1 -> z3, y-2 -> z4,

In [72]:
```
using JuMP, Clp

m = Model(solver=ClpSolver())
@variable(m, u >= 0 )
@variable(m, v >= 0 )
@variable(m, w >= 0 )
@variable(m, x >= 0 )
@variable(m, y >= 0 )
@variable(m, s1 >= 0 )
@variable(m, s2 >= 0 )
@variable(m, s3 >= 0 )
@variable(m, s4 >= 0 )
@variable(m, s5 >= 0 )
@constraint(m, w - 6 + s1 == 0)
@constraint(m, x - 6 + s2 == 0)
@constraint(m, y - 4 + s3 == 0)
@constraint(m, (u-v) - 6(w-1) + (x-1) - (y-2) -3 + s4 == 0 )
@constraint(m, 7(w-1) + (y-2) == 5 )
@constraint(m, (x-1) + (y-2) -2 + s5 == 0 )
@objective(m, Min, -3(u-v) + (w-1) )

status = @time for i = 1:10 solve(m) end

println(m)
println(status)
println()
println("z1 = ", getvalue(u-v) )
println("z2 = ", getvalue(w-1) )
println("z3 = ", getvalue(x-1) )
println("z4 = ", getvalue(y-2) )
println("objective = ", -getobjectivevalue(m) )
```

```
  0.012899 seconds (390 allocations: 33.547 KiB)
Min -3 u + 3 v + w - 1
Subject to
 w + s1 = 6
 x + s2 = 6
 y + s3 = 4
 u - v - 6 w + x - y + s4 = -4
 7 w + y = 14
 x + y + s5 = 5
 u ≥ 0
 v ≥ 0
 w ≥ 0
 x ≥ 0
 y ≥ 0
 s1 ≥ 0
 s2 ≥ 0
 s3 ≥ 0
 s4 ≥ 0
 s5 ≥ 0

nothing

z1 = 8.571428571428571
z2 = 0.4285714285714286
z3 = -1.0
z4 = 2.0
objective = 25.28571428571429
```

In [73]:
```
using JuMP, Clp

m = Model(solver=ClpSolver())
@variable(m, z1 )
@variable(m, -1 <= z2 <= 5 )
@variable(m, -1 <= z3 <= 5 )
@variable(m, -2 <= z4 <= 2 )
@constraint(m,z1 - 6z2 + z3 - z4 <= 3 )
@constraint(m,7z2 + z4 == 5 )
@constraint(m,z3 + z4 <= 2 )
@objective(m, Max, 3z1 - z2 )

status = @time for i = 1:10 solve(m) end

println(m)
println(status)
println()
println("z1 = ", getvalue(z1) )
println("z2 = ", getvalue(z2) )
println("z3 = ", getvalue(z3) )
println("z4 = ", getvalue(z4) )
println("objective = ", getobjectivevalue(m) )
```

```
  0.008903 seconds (390 allocations: 27.938 KiB)
Max 3 z1 - z2
Subject to
 z1 - 6 z2 + z3 - z4 ≤ 3
 7 z2 + z4 = 5
 z3 + z4 ≤ 2
 z1
 -1 ≤ z2 ≤ 5
 -1 ≤ z3 ≤ 5
 -2 ≤ z4 ≤ 2

nothing

z1 = 8.571428571428571
z2 = 0.42857142857142855
z3 = -1.0
z4 = 2.0
objective = 25.28571428571429
```

Q3.

In [74]:
```
using JuMP, Clp

m = Model(solver=ClpSolver())
@variable(m,  2    <= x <=  3    )
@variable(m,  0.4  <= y <=  0.6  )
@variable(m,  1.2  <= z <=  1.65 )
@variable(m, 94.75 <= r <= 96.4  )
@variable(m, 0 <= I1 <= 1 )
@variable(m, 0 <= I2 <= 1 )
@variable(m, 0 <= I3 <= 1 )
@variable(m, 0 <= C1 <= 1 )
@variable(m, 0 <= C2 <= 1 )
@variable(m, 0 <= A1 <= 1 )
@variable(m, 0 <= A2 <= 1 )
@constraint(m, x + y + z + r == 100 )
@constraint(m, 2.5I1 + 3I2 - x == 0 )
@constraint(m, 1.3I1 + 0.8I2 + 4C2 + 1.2A1 - z == 0 )
@constraint(m, 0.3I3 + 90C1 + 96C2 + 0.4A1 + 0.6A2 - y == 0 )
@constraint(m, 400I1 + 300I2 + 600I3 + 500C1 + 200C2 + 300A1 + 250A2 ==
500 )
@objective(m, Min, 200I1 + 250I2 + 150I3 + 220C1 + 240C2 + 200A1 + 165A2
 )

status = @time for i = 1:10 solve(m) end

println(m)
println(status)
println()
println("Carbon = ", getvalue(x) )
println("Copper = ", getvalue(y) )
println("Manganese = ", getvalue(z) )
println("Iron alloy 1 = ", getvalue(I1) )
println("Iron alloy 2 = ", getvalue(I2) )
println("Iron alloy 3 = ", getvalue(I3) )
println("Copper 1 = ", getvalue(C1) )
println("Copper 2 = ", getvalue(C2) )
println("Aluminum 1 = ", getvalue(A1) )
println("Aluminum 2 = ", getvalue(A2) )
println("objective = ", getobjectivevalue(m) )
```

```
  0.005338 seconds (390 allocations: 35.688 KiB)
Min 200 I1 + 250 I2 + 150 I3 + 220 C1 + 240 C2 + 200 A1 + 165 A2
Subject to
 x + y + z + r = 100
 2.5 I1 + 3 I2 - x = 0
 1.3 I1 + 0.8 I2 + 4 C2 + 1.2 A1 - z = 0
 0.3 I3 + 90 C1 + 96 C2 + 0.4 A1 + 0.6 A2 - y = 0
 400 I1 + 300 I2 + 600 I3 + 500 C1 + 200 C2 + 300 A1 + 250 A2 = 500
 2 ≤ x ≤ 3
 0.4 ≤ y ≤ 0.6
 1.2 ≤ z ≤ 1.65
 94.75 ≤ r ≤ 96.4
 0 ≤ I1 ≤ 1
 0 ≤ I2 ≤ 1
 0 ≤ I3 ≤ 1
 0 ≤ C1 ≤ 1
 0 ≤ C2 ≤ 1
 0 ≤ A1 ≤ 1
 0 ≤ A2 ≤ 1

nothing

Carbon = 2.265083286841476
Copper = 0.6000000000000001
Manganese = 1.2
Iron alloy 1 = 0.9060333147365904
Iron alloy 2 = 0.0
Iron alloy 3 = 0.22746473260540379
Copper 1 = 0.0
Copper 2 = 0.005539172710608113
Aluminum 1 = 0.0
Aluminum 2 = 0.0
objective = 216.6557742886746
```

Q4.

Stigler's diet costed 39.93 dollars per year but according to the current solution it would cost 39.66173154546625 dollars per year. Wheat Flour (Enriched),Liver (Beef),Cabbage, Spinach, Navy Beans, Dried form the optimal diet.

For vegetarian diet the minimum annual cost for optminal diet would be 39.79866435040897 dollars. Navy Beans, Dried, Spinach, Cabbage, Evaporated Milk (can), Wheat Flour (Enriched) would make up the optimal diet.

In [75]:
```
# STARTER CODE FOR STIGLER'S DIET PROBLEM
using NamedArrays

# import Stigler's data set
raw = readcsv("stigler.csv")
(m,n) = size(raw)

n_nutrients = 2:n      # columns containing nutrients
n_foods = 3:m          # rows containing food names

nutrients = raw[1,n_nutrients][:]   # the list of nutrients (convert to
 1-D array)
foods = raw[n_foods,1][:]           # the list of foods (convert to 1-D
 array)

veg_food = foods[:]
filter!(e->e∉["Pork and Beans (can)","Crisco","Lard","Sirloin Steak","Ro
und Steak","Rib Roast","Chuck Roast","Plate","Liver (Beef)","Leg of Lam
b","Lamb Chops (Rib)","Pork Chops","Pork Loin Roast","Bacon","Ham, smoke
d","Salt Pork","Roasting Chicken","Veal Cutlets","Salmon, Pink (can)"],v
eg_food)

# lower[i] is the minimum daily requirement of nutrient i.
lower = Dict( zip(nutrients,raw[2,n_nutrients]) )

# data[f,i] is the amount of nutrient i contained in food f.
data = NamedArray( raw[n_foods,n_nutrients], (foods,nutrients), ("foods"
,"nutrients") );
```

In [76]:
```
using JuMP, Clp

m = Model(solver=ClpSolver())

@variable(m, 0 <= food_perc[foods] <=1 )

@expression(m, total_cost, sum(food_perc[i] for i in foods))

@constraint(m, constr[j in nutrients], sum(data[i,j] * food_perc[i] for
i in foods ) >= lower[j] )

@objective(m, Min, total_cost)

solve(m)
# println(getvalue(food_perc))
println(getvalue(365*total_cost))

# println(lower)
# println(final_nutrients)
# println(constr)
# println(lower["Calories (1000)"])
# println(data[2,1])
# println(data)
```

```
39.66173154546625
```

In [77]:
```
using JuMP, Clp

m = Model(solver=ClpSolver())

@variable(m, 0 <= food_perc[veg_food] <=1 )

@expression(m, total_cost, sum(food_perc[i] for i in veg_food))

@constraint(m, constr[j in nutrients], sum(data[i,j] * food_perc[i] for
i in veg_food ) >= lower[j] )

@objective(m, Min, total_cost)

solve(m)
# println(getvalue(food_perc))
println(getvalue(365*total_cost))

# println(lower)
# println(final_nutrients)
# println(constr)
# println(lower["Calories (1000)"])
# println(data[2,1])
# println(data)
```

```
39.79866435040897
```