In [1]:
```
using DynamicPolynomials,LinearAlgebra
using NBInclude

@nbinclude("MinMaxonSet.ipynb")

using JuMP
using MadNLP
import HiGHS
```

Input

- Matrix \$H\$ and vector \$d\$ so that \$\mathcal{X} = \lbrace x \in \mathbb{R}^n | Ax=c \rbrace\$
- Signomial defined by Matrix \$A^{m \times n}\$ and coeffiecients \$c \in \mathbb{R}^m\$ ## Example

In [2]:
```
c = [1,-1,-1]
A=[3//2 0;0 1;0 -1]
```

Out[2]:
```
3×2 Matrix{Rational{Int64}}:
 3//2   0//1
 0//1   1//1
 0//1  -1//1
```

In [3]:
```
H=[1 0;-1 0;0 1;0 -1]
d=[1,-1,1,1]
```

Out[3]:
```
4-element Vector{Int64}:
  1
 -1
  1
  1
```

At first we need to calculate the leftinverse of A

In [4]:
```
function rpinv(A) ## rational leftinverse
    rA = Rational.(A)
    B = (transpose(rA)*rA)\transpose(rA)
    return B
end
```

Out[4]:
```
rpinv (generic function with 1 method)
```

In [5]:
```
function getSupport(B,d)
    Supp=[]
    nvars=size(B,2)
    # print(nvars)
    for (i,row) in enumerate(eachrow(B))
        P=[]
        N=[]
        denom=lcm(denominator.(row))
        for (j,ele) in enumerate(row)
            if ele >= 0
                push!(P,(Int(ele*denom),j))
            else
                push!(N,(Int(-ele*denom),j))
            end
        end
```

```
        pdegree=sum(Int[x[1] for x in P])
        ndegree=sum(Int[x[1] for x in N])
        if pdegree>ndegree
            push!(N,(pdegree-ndegree,nvars+1))
        elseif pdegree<ndegree
            push!(P,(ndegree-pdegree,nvars+1))
        end
        push!(Supp,(P,N,denom*d[i]))
    end
    return (Supp,nvars)
end
```

Out[5]:
```
getSupport (generic function with 1 method)
```

In [6]:
```
function getSASet(Supp,nvars)
    F=[]
    @polyvar x[1:nvars+1]
    for ele in Supp
        s=ele
        P=s[1]
        N=s[2]
        dᵢ=s[3]
        f=x[1]*0
        ## Add positive part
        p=x[1]*0+1
        for pos in P
            p=p*x[pos[2]]^pos[1]
        end
        f=f-p
        ##Add negative part
        p=x[1]*0+ℯ^dᵢ
        for neg in N
            p=p*x[neg[2]]^neg[1]
        end
        f=f+p
        push!(F,f)
    end
    return (F,x)
end
```

Out[6]:
```
getSASet (generic function with 1 method)
```

In [7]:
```
function SASbyMatrix(A,H,c,d)
    ##ToDo Check Dimensions
    L=rpinv(A)
    B=H*L
    Supp,nvars=getSupport(B,d)
    # [println(u) for u in eachrow(Supp)]
    return getSASet(Supp,nvars)
end
```

Out[7]:
```
SASbyMatrix (generic function with 1 method)
```

In [8]:
```
# Support ist of the Form: every Row is a constraint polynomials with to vectors po
```

In [9]:
```
SASbyMatrix(A,H,c,d)
```

```
Any[(Any[(2, 1), (0, 2), (0, 3)], Any[(2, 4)], 3)]
Any[(Any[(0, 2), (0, 3), (2, 4)], Any[(2, 1)], -3)]
Any[(Any[(0, 1), (1, 2)], Any[(1, 3)], 2)]
Any[(Any[(0, 1), (1, 3)], Any[(1, 2)], 2)]
```

Out[9]:
```
(Any[-x₁² + 20.085536923187668x₄², 0.0497870683678639 44x₁² - x₄², -x₂ + 7.38905609893065x₃, 7.38905609893065x₂ - x₃], PolyVar{true}[x₁, x₂, x₃, x₄])
```

Add Redundant Constraints

In [10]:
```
function getRedCons(A,H,d,var)
    R=[]
    # Start with upper bounds
    C = findMinOnSet(A,H,d)
    nvars = size(C,1) #Number of  Variables
    for i in 1:nvars
        fun = var[i]-ℯ^C[i][1]*var[nvars+1]
        push!(R,fun)
    end

    # Add lower bounds
     D = findMaxOnSet(A,H,d)
    for i in 1:size(D,1)
        fun = ℯ^D[i][1]*var[nvars+1]-var[i]
        push!(R,fun)
    end
    return (R,nvars)
end
```

Out[10]:
```
getRedCons (generic function with 1 method)
```

In [11]:
```
function getallConstraints(A,H,c,d)
    SAS,var = SASbyMatrix(A,H,c,d)
    RC,nvars = getRedCons(A,H,d,var)
    Sol = vcat(SAS, RC)
    # Add Homogenization Constraint
    push!(Sol,var[nvars+1])
    # Objective Function
    f= sum(c.*var[1:nvars])
    return (f,Vector(Sol))

end
f,Sol = getallConstraints(A,H,c,d)
variables([Sol...,f])
```

```
Any[(Any[(2, 1), (0, 2), (0, 3)], Any[(2, 4)], 3)]
Any[(Any[(0, 2), (0, 3), (2, 4)], Any[(2, 1)], -3)]
Any[(Any[(0, 1), (1, 2)], Any[(1, 3)], 2)]
Any[(Any[(0, 1), (1, 3)], Any[(1, 2)], 2)]
```

Out[11]:
```
4-element Vector{PolyVar{true}}:
 x₁
 x₂
 x₃
 x₄
```

In [12]:
```
f
```

Out[12]: $$ x_{1} - x_{2} - x_{3} $$

In [13]:
```
getallConstraints(A,H,c,d)
```

```
Any[(Any[(2, 1), (0, 2), (0, 3)], Any[(2, 4)], 3)]
Any[(Any[(0, 2), (0, 3), (2, 4)], Any[(2, 1)], -3)]
Any[(Any[(0, 1), (1, 2)], Any[(1, 3)], 2)]
Any[(Any[(0, 1), (1, 3)], Any[(1, 2)], 2)]
```

Out[13]: $(x_1 - x_2 - x_3, \mathrm{Any}[-x_1^2 + 20.085536923187668x_4^2, 0.049787068367863944x_1^2 - x_4^2, -x_2 + 7.38905609893065x_3, 7.38905609893065x_2 - x_3, x_1 - 4.4816890703380645x_4, x_2 - 0.36787944117144233x_4, x_3 - 0.36787944117144233x_4, -x_1 + 4.4816890703380645x_4, -x_2 + 2.718281828459045x_4, -x_3 + 2.718281828459045x_4, x_4])$

In [ ]: