In [1]:

```
using Gurobi, JuMP, PyPlot
```

Solution 1: The Huber loss.

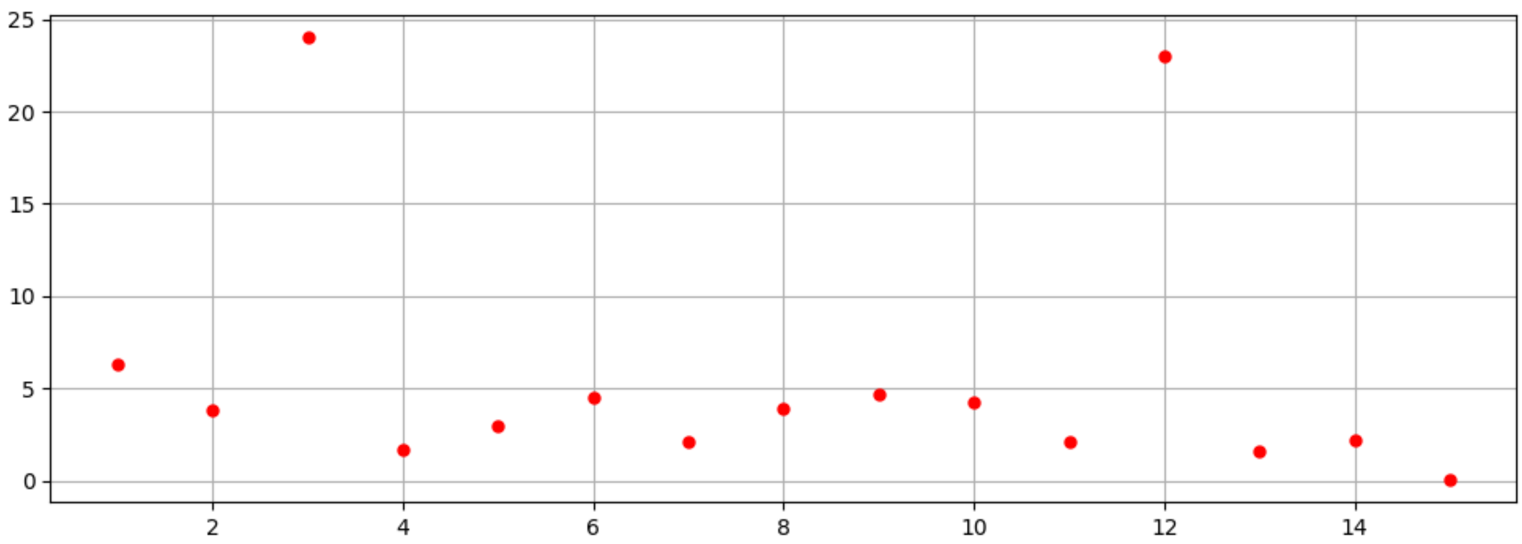
In statistics, we frequently encounter data sets containing outliers, which are bad data points arising from experimental error or abnormally high noise. Consider for example the following data set consisting of 15 pairs (x, y).

Solution 1a: Compute the best linear fit to the data using an l2 cost (least squares). In other words, we are looking for the a and b that minimize the expression:

$$l_2 cost: \sum_{i=1}^{15} (y_i - a x_i - b)^2$$

In [2]:

```
# define (x,y) coordinates of the points
x = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ]
y = [ 6.31, 3.78, 24, 1.71, 2.99, 4.53, 2.11, 3.88, 4.67, 4.25, 2.06, 23, 1.58,
2.17, 0.02 ]

using PyPlot
figure(figsize=(12,4))
plot(x,y,"r.", markersize=10)
grid("on")

len_x = length(x)
```



Out[2]:

15

In [3]:

```
m1a = Model(solver=GurobiSolver(OutputFlag=0))
@variable(m1a, a)
@variable(m1a, b)
@objective(m1a, Min, sum( (y[i] - a*x[i] - b).^2 for i in 1:length(x) ))
status = solve(m1a)
val_a = getvalue(a)
val_b = getvalue(b)
println(status)
println(val_a)
println(val_b)
```

```
Optimal
-0.29078571428551947
8.130285714279665
```

In [6]:

```
# Calculating values of y based on the values of a and b that we have learned fr
om above and using the x coordinate
# values
y1 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
for i = 1:length(x)
    y1[i] += (val_a * x[i]) + val_b
end
```

In [7]:

```
# Removing the outliers this time.
x2 = [ 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15 ]
y2 = [ 6.31, 3.78, 1.71, 2.99, 4.53, 2.11, 3.88, 4.67, 4.25, 2.06, 1.58, 2.17, 0
.02 ]

m1a1 = Model(solver=GurobiSolver(OutputFlag=0))
@variable(m1a1, a)
@variable(m1a1, b)
@objective(m1a1, Min, sum( (y2[i] - a*x2[i] - b).^2 for i in 1:length(x2) ))
status = solve(m1a1)
val_a1 = getvalue(a)
val_b1 = getvalue(b)
println(status)
println(val_a1)
println(val_b1)
```

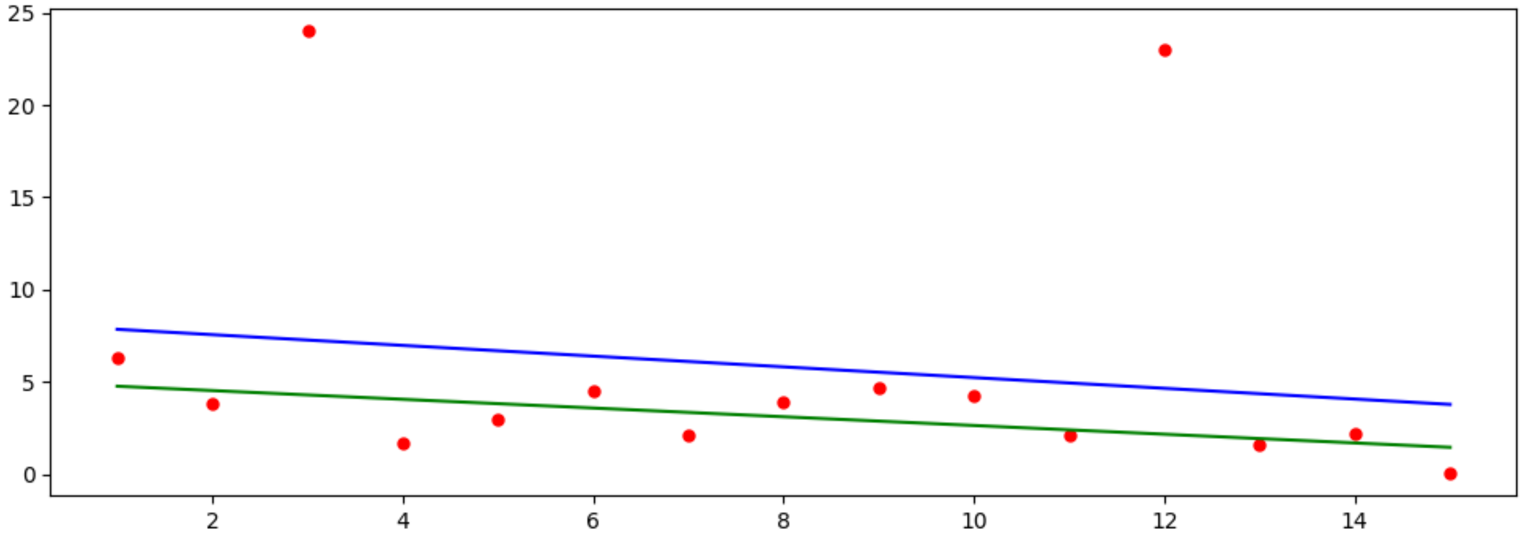
```
Optimal
-0.23648422408233874
4.9916033483557305
```

In [8]:

```
# Getting the y values after removing the outlier and based on new values of a a
nd b (without outliers)
x2 = [ 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15 ]
y2 = [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ]
for i = 1:length(x2)
    y2[i] += (val_a1 * x2[i]) + val_b1
end
```

In [10]:

```
figure(figsize=(12,4))
plot( x, y, "r.", markersize=10)
plot( x, y1, "b-")
plot( x2, y2, "g-")
println("Blue line is when we consider outliers 3 and 12 points")
println("Green line is when we DON'T consider outliers 3 and 12 points")
```




```
Blue line is when we consider outliers 3 and 12 points
Green line is when we DON'T consider outliers 3 and 12 points
```

Explanation: We can see that when outliers are not considered the plot is more close to the rest of the points (Green line) than when we consider the outliers (blue line). This is because the best linear fit using l2 (least squares) in blue line is trying to consider the points which are far above as well.

Solution 1b: It's not always practical to remove outliers from the data manually, so we'll investigate ways of automatically dealing with outliers by changing our cost function. Find the best linear fit again (including the outliers), but this time use the l1 cost function:

$$l_1 cost : \sum_{i=1}^{15} |y_i - ax_i - b|$$

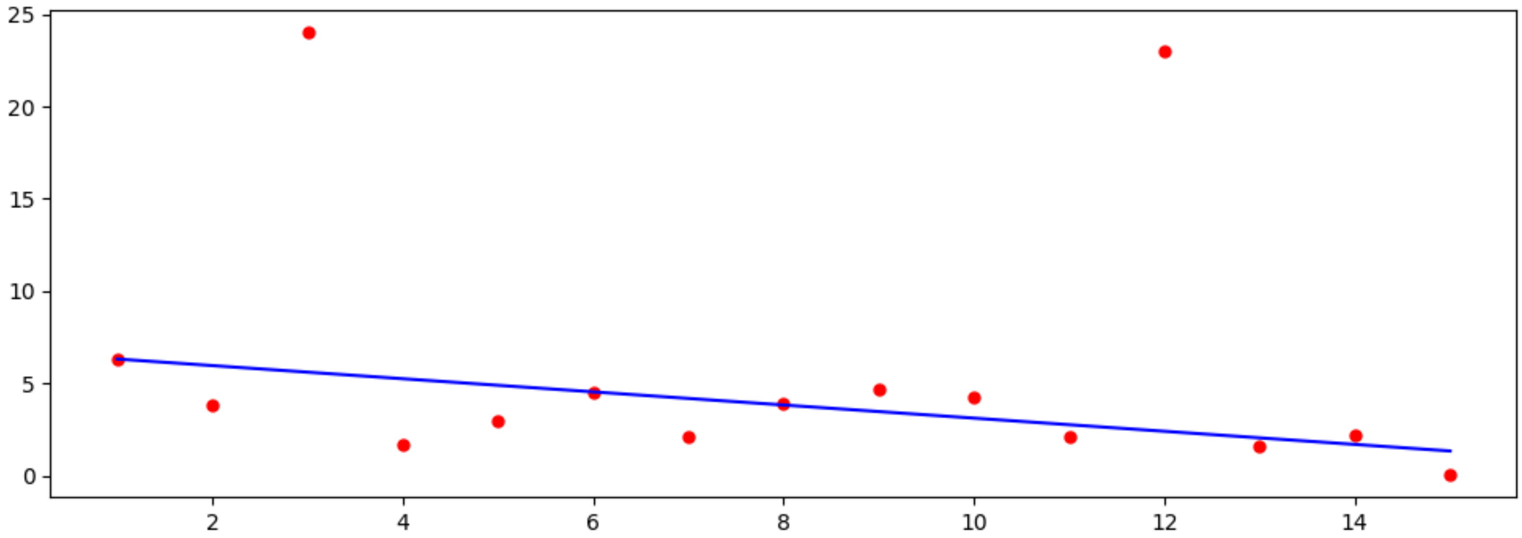
In [11]:

```
using JuMP, ECOS
x = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ]
y = [ 6.31, 3.78, 24, 1.71, 2.99, 4.53, 2.11, 3.88, 4.67, 4.25, 2.06, 23, 1.58,
2.17, 0.02 ]

m1b = Model(solver=GurobiSolver(OutputFlag=0))
@variable(m1b, a1)
@variable(m1b, b1)
@variable(m1b, t[1:length(x)])
@expression(m1b, S1[i=1:length(x)], (y[i] - a1*x[i] - b1))

@constraint(m1b, S1 .<= t)
@constraint(m1b, S1 .>= -t)

@objective(m1b, Min, sum(t))

status = solve(m1b)
val_a2 = getvalue(a1)
val_b2 = getvalue(b1)
println(status)
println("Value of a: ", val_a2)
println("Value of b: ", val_b2)
```

```
Optimal
Value of a: -0.35599999999999987
Value of b: 6.6659999999999995
```

In [12]:

```
y1b = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
]
for i = 1:length(x)
    y1b[i] += (val_a2 * x[i]) + val_b2
end
figure(figsize=(12,4))
plot( x, y, "r.", markersize=10)
plot( x, y1b, "b-")
```



Out[12]:

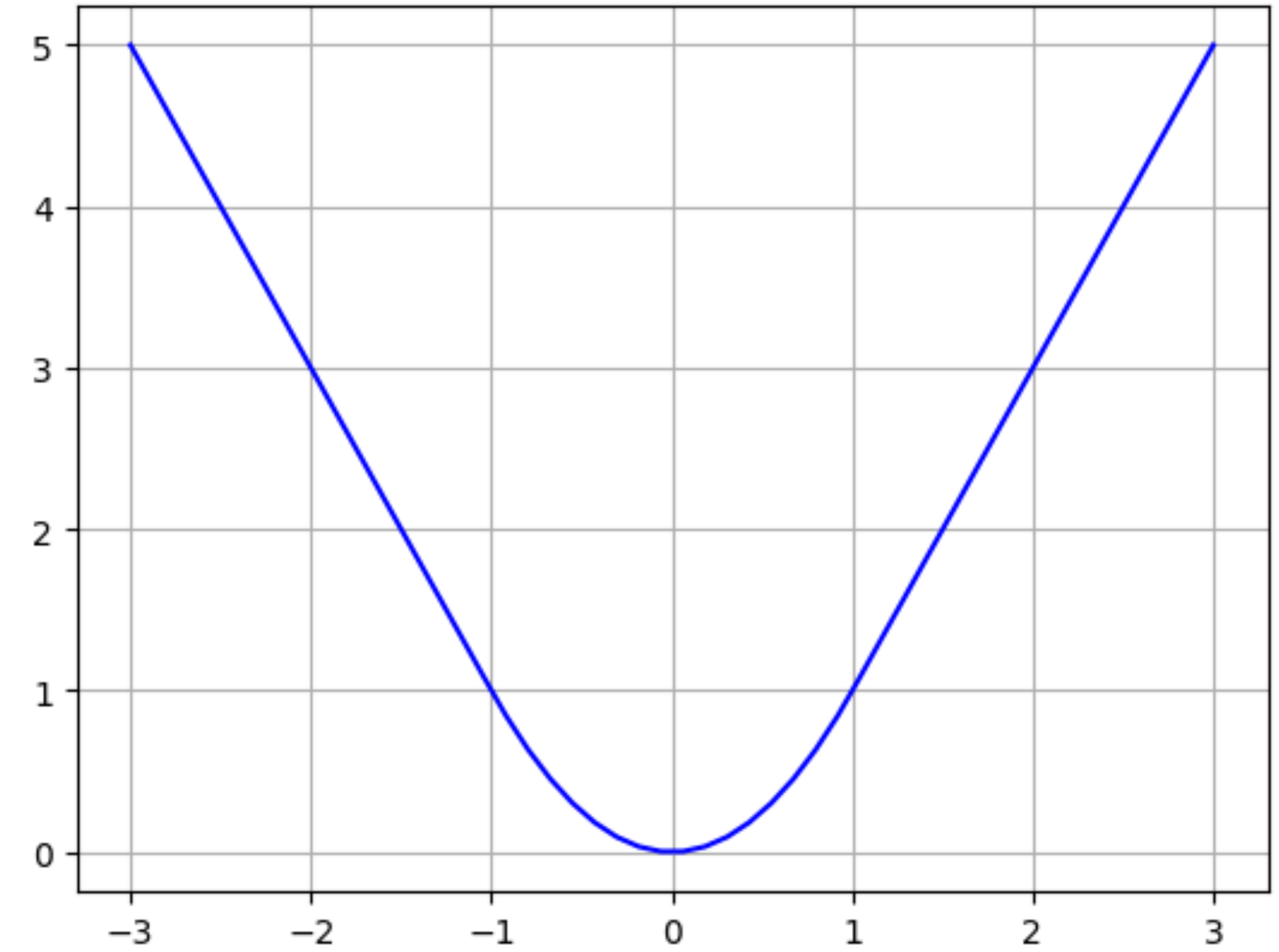
```
1-element Array{Any,1}:
 PyObject <matplotlib.lines.Line2D object at 0x326409b90>
```

Explanation: Using the cost function l1 and considering outliers this time we see the above plot. We can see that the above linear fit doesn't do better than least squares that was used in Solution 1a. This plot seems to be not considering outliers because of the defined l1 cost function.

<u>Solution 1c:</u> Another approach is to use an l2 penalty for points that are close to the line but an l1 penalty for points that are far away. Specifically, we'll use something called the Huber loss.

In [229]:

```
function huber_loss(x)
    M = 1
    m1c = Model(solver=GurobiSolver(OutputFlag=0))
    @variable(m1c, v >= 0)
    @variable(m1c, w <= M) # Given M = 1

    @constraint(m1c, x <= (w + v))
    @constraint(m1c, x >= -(w + v))

    @objective(m1c, Min, (w^2 + 2 * M * v))
    solve(m1c)
    return getobjectivevalue(m1c)
end

x = linspace(-3, 3)
y = [huber_loss(items) for items in x]

grid("on")
plot( x, y, "b", label="Huber Loss Sol 1c")
```




```
WARNING: Method definition huber_loss(Any) in module Main at In[228]
:2 overwritten at In[229]:2.
```

Out[229]:

```
1-element Array{Any,1}:
 PyObject <matplotlib.lines.Line2D object at 0x3294aec10>
```

In [252]:

```
M = 1
m1c1 = Model(solver=GurobiSolver(OutputFlag=0))
@variable(m1c1, a1c)
@variable(m1c1, b1c)
@variable(m1c1, w[1:len_x] <= 1)
@variable(m1c1, v[1:len_x] >= 0)

@constraint(m1c1, y - a1c * x - b1c .<= (w + v))
@constraint(m1c1, y - a1c * x - b1c .>= -(w + v))

@objective(m1c1, Min, sum(w.^2 + 2 * M * v))
status = solve(m1c1)
println(status)

aopt = getvalue(a1c)
bopt = getvalue(b1c)

println("Value of a: ", aopt)
println("Value of b: ", bopt)
```

```
Optimal
Value of a: -0.2811079944792559
Value of b: 5.738120618207082
```

In [251]:

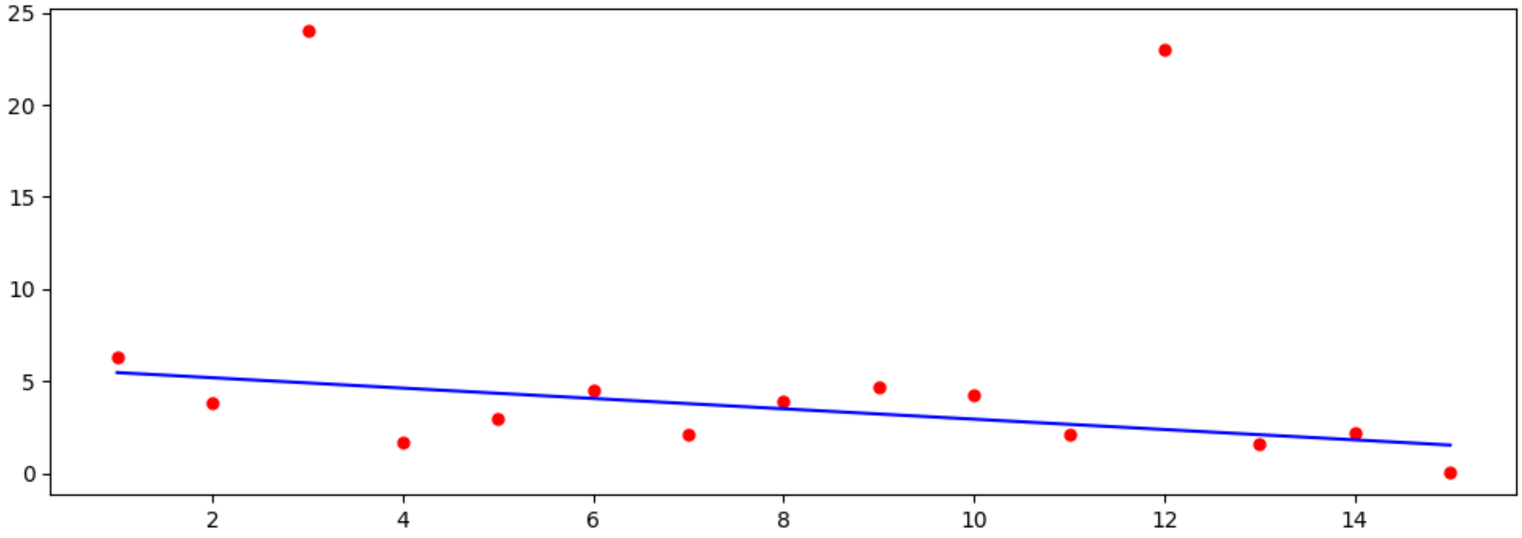
```
solve(m1c1)
```

Out[251]:

```
:Optimal
```

In [254]:

```
x = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ]
y1c = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
]
for i = 1:length(x)
    y1c[i] += (aopt * x[i]) + bopt
end
figure(figsize=(12,4))
plot( x, y, "r.", markersize=10)
plot( x, y1c, "b-")
```



Out[254]:

```
1-element Array{Any,1}:
 PyObject <matplotlib.lines.Line2D object at 0x323440710>
```

In [ ]:

Solution 2 (a)

following constraint are equivalent

$$w^T w \leq xy, \quad x \geq 0, y \geq 0 \iff \left\| \begin{bmatrix} 2w \\ x-y \end{bmatrix} \right\| \leq x+y$$

and we are asked to express

$$t(a^T x + b) \geq 1 \text{ as SOCP}$$

Creating & substituting mapping

$$\boxed{\begin{aligned} y &= a^T x + b \\ x &= t \end{aligned}}$$

we see that for $\boxed{w = 1}$ we get

$$\text{eq ① } \text{-----} \left\| \begin{bmatrix} 2(1) \\ t - (a^T x + b) \end{bmatrix} \right\| \leq t + (a^T x + b)$$

where $a^T x + b \geq 0$

Re-arranging eq ① in known format

$$\left\| \begin{bmatrix} 0 \\ -a^T \end{bmatrix} x + \begin{bmatrix} 2 \\ t-b \end{bmatrix} \right\| \leq (a)^T x + (b+t)$$

$$\left. \begin{aligned} A &= \begin{bmatrix} 0 \\ -a^T \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ t-b \end{bmatrix}, \quad c = a, \quad x = x \\ d &= t + b \end{aligned} \right\} \underline{\underline{\text{Soln}}}$$

Soln 2(b)

Hyperbolic optimization problem:

$$\min_{x} \sum_{i=1}^{p} 1/(a_i^T x + b_i)$$

$$\text{s.t} \quad a_i^T x + b_i > 0, \quad i = 1, \cdots, p$$

$$c_j^T x + d_j \geq 0 \quad j = 1, \ldots, q$$

Writing this as SOCP:

If we write $\dfrac{1}{(a_i^T x + b_i)}$ as z_i

we will have

$$\min_{x,z} \sum_{i=1}^{p} z_i$$

but now we need to add constraint such that

$$\frac{1}{(a_i^T x + b_i)} \leq z_i$$

from soln 2(a) we know how to write above as SOCP so, final soln can be.

$$\min_{z} \sum_{i=1}^{p} z_i$$

$$\text{s.t} \quad \left\| \begin{bmatrix} 0 \\ -a_i^T \end{bmatrix} x + \begin{bmatrix} 2 \\ t_i - b_i \end{bmatrix} \right\| \leq a_i^T x + b_i + z_i$$

$$a_i^T x + b_i > 0 \quad ; i = 1, \ldots, p$$

$$c_j^T x + d_j \geq 0 \quad ; j = 1, \ldots q$$

Solution 3(a) $\quad \max \ \alpha_4 T r^2$

subject to: $\quad \dfrac{\alpha_1 T r}{w} + \alpha_2 r + \alpha_3 r w \leq C_{max}$

$T_{min} \leq T \leq T_{max}$

$r_{min} \leq r \leq r_{max}$

$w_{min} \leq w \leq w_{max}$

$w \leq 0.1 r$

Expressing this problem as geometric program, & convert into a convex optimization problem

$$\min \ (\alpha_4 T r^2)^{-1} \quad \text{———} \quad (1)$$

s.t.

$$\left.\begin{array}{ll} T_{min} T^{-1} \leq 1 & (1/T_{max}) \, T \leq 1 \\ r_{min} r^{-1} \leq 1 & (1/r_{max}) \, r \leq 1 \\ w_{min} w^{-1} \leq 1 & (1/w_{max}) \, w \leq 1 \\ 10 w r^{-1} \leq 1 & \dfrac{\alpha_1 T r}{w} + \alpha_2 r + \alpha_3 r w \leq C_{max} \end{array}\right\} \quad (2)$$

Taking log(exp(log())) of everything & substituting

$$x := \log T, \quad y := \log r, \quad z := \log w$$

(1) becomes :-

$$\log(\exp(\log(\alpha_4^{-1} . T^{-1} r^{-2})))$$

$$\log(\exp(\log(\alpha_4^{-1}) + \log(T^{-1}) + \log(r^{-2})))$$

$$\log(\exp(-\log(\alpha_4) - \log(T) \cancel{+} -2\log(r)))$$

$$\boxed{-\log \alpha_4 - x - 2y}$$

② becomes

$$\log(\exp(\log(T_{min}\, T^{-1}))) \leq 0$$

$$\log(\exp(\log(T_{min}) - \log(T)) \leq 0$$

$$\log T_{min} - x \leq 0$$

Similarly $-\log T_{max} + x \leq 0$

so, $\boxed{\log T_{min} \leq x \leq \log T_{max}}$

Similarly,

$$\boxed{\begin{aligned} \log r_{min} &\leq y \leq \log r_{max} \\ \log w_{min} &\leq z \leq \log w_{max} \end{aligned}}$$

Also, $\log(\exp(\log(10\, w\, r^{-1}))) \leq 0$

$$\boxed{z - y \leq -\log(10)}$$

And finally

$$\log\left(\exp\left(\log\left(\frac{\alpha_1 T r w^{-1}}{c_{max}} + \frac{\alpha_2 r}{c_{max}} + \frac{\alpha_3 r w}{c_{max}}\right)\right)\right) \leq 0$$

$$\log\left(e^{\log\left(\frac{\alpha_1}{c_{max}}\right) + x + y - z} + e^{\log\left(\frac{\alpha_2}{c_{max}}\right) + y} + e^{\log\left(\frac{\alpha_3}{c_{max}}\right) + y + z}\right) \leq 0$$

final solu^n

$$\min_{x,y,z} \quad -\log(\alpha_4) - x - 2y$$

$$\text{s.t.} \quad \log\left(e^{\log(\alpha_1/C_{max}) + x + y - z} + e^{\log\left(\frac{\alpha_2}{C_{max}}\right) + y} + e^{\log\left(\frac{\alpha_3}{C_{max}}\right) + y + z}\right) \leq 0$$

$$z - y \leq -\log(10)$$

$$\log T_{min} \leq x \leq \log T_{max}$$
$$\log r_{min} \leq y \leq \log r_{max}$$
$$\log w_{min} \leq z \leq \log w_{max}$$

<u>Solution 3b:</u> Consider a simple instance of this problem, where Cmax = 500 and α1 = α2 = α3 = α4 = 1. Also assume for simplicity that each variable has a lower bound of zero and no upper bound. Solve this problem using JuMP. Use the Mosek solver and the command @NLconstraint(...) to specify nonlinear constraints such as log-sum-exp functions. Note: Mosek can solve general convex optimization problems! What is the optimal T, r, and w?

In [2]:

```julia
using JuMP, Mosek
```

In [3]:

```julia
Cmax = 500
a1 = 1
a2 = 1
a3 = 1
a4 = 1
m3b = Model(solver=MosekSolver(LOG=0))
@variable(m3b, x)
@variable(m3b, y)
@variable(m3b, z)
@NLconstraint(m3b, log(exp(log(a1/Cmax)+x+y-z) + exp(log(a2/Cmax)+y) + exp(log(a
3/Cmax)+y+z)) <= 0)
@constraint(m3b, log(10) + z - y <= 0)
@objective(m3b, Min, -log(1) - x - 2y)
solve(m3b)
```

Out[3]:

```
:Optimal
```

In [5]:

```julia
m3b
```

Out[5]:

$$\begin{aligned} \min \quad & -x-2y \\ \text{Subject to} \quad & z-y \leq -2.302585092994046 \\ & log(exp((log(1.0/500.0)+x+y)-z)+exp(log(1.0/500.0)+y)+exp(lo \\ & x free \\ & y free \\ & z free \end{aligned}$$

In [4]:

```
println("Value of x: ", getvalue(x))
println("Value of y: ", getvalue(y))
println("Value of z: ", getvalue(z))

println("--------------------------")

println("Value of T: ", exp(getvalue(x)))
println("Value of r: ", exp(getvalue(y)))
println("Value of w: ", exp(getvalue(z)))
```

```
Value of x: 3.1713737927364014
Value of y: 3.8370936491865004
Value of z: 1.534507498647674
--------------------------
Value of T: 23.840213381810717
Value of r: 46.39045139651206
Value of w: 4.639040233655826
```

In [ ]:

```

```