Homework 6 Question 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).

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

The y values corresponding to $x = 3$ and $x = 12$ are outliers because they are far outside the expected range of values for the experiment.

a) Compute the best linear fit to the data using an ℓ_2 cost (least squares). In other words, we are looking for the a and b that minimize the expression:

$$\ell_2 \; cost: \sum_{i=1}^{15}(y_i - ax_i - b)^2$$

Repeat the linear fit computation but this time exclude the outliers from your data set. On a single plot, show the data points and both linear fits. Explain the difference between both fits.

In [1]:
```
x = linspace(1,15,15)
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]';
```

Least Squares

In [2]:
```
# order of polynomial to use
k=1
# fit using a function of the form f(x) = u1 x^k + u2 x^(k-1) + ... + uk x + u{k+1}
n = length(x)
A = zeros(n,k+1)
for i = 1:n
    for j = 1:k+1
        A[i,j] = x[i]^(k+1-j)
    end
end
```

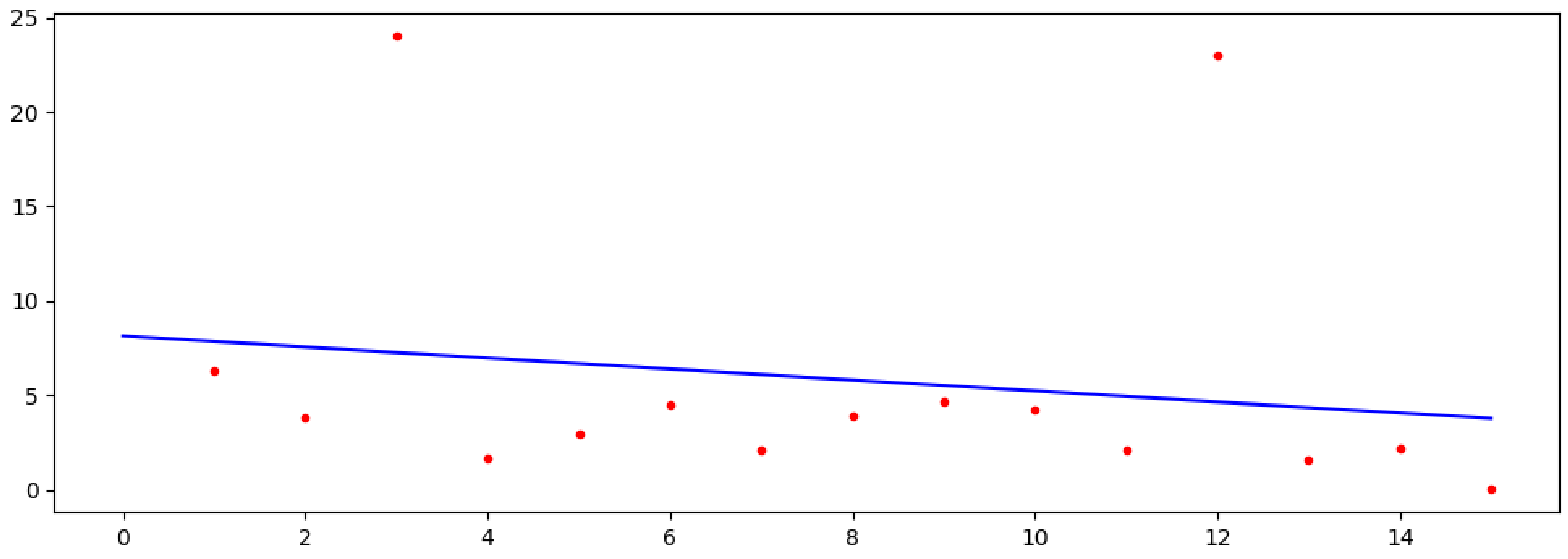
In [3]:
```
using JuMP, Gurobi
import JuMP: GenericAffExpr

# helper function defined to model linear constraints for absolute value in JuMP for array variables
function abs_array{V<:GenericAffExpr}(v::Array{V})
    m = first(first(v).vars).m
    @variable(m, aux[1:length(v)] >= 0)
    @constraint(m, aux .>= v)
    @constraint(m, aux .>= -v)
    return aux
end;
```

In [4]:
```

m = Model(solver=GurobiSolver(OutputFlag=0))
@variable(m, u[1:k+1])
@objective(m, Min, sum((y - A*u).^2))

status = solve(m)
uopt = getvalue(u)
println(status)
println("L2 Error: ",norm(y - A*uopt),
    " Objective value: ",getobjectivevalue(m),", Parameters learnt: ", uopt)
```

```
Optimal
L2 Error: 27.08033949460119 Objective value: 733.3447871428841, Parameters learnt: [-0.290786,8.1302
9]
```

Plot

In [5]:
```
using PyPlot
npts = 100
xfine = linspace(0,15,npts)
ffine = ones(npts)
for j = 1:k
    ffine = [ffine.*xfine ones(npts)]
end
yfine = ffine * uopt

figure(figsize=(12,4))
plot( x, y, "r.")
plot( xfine, yfine, "b-");
```



Cleaning the data by removing the enrty of $x = 3$ and $x = 12$

In [6]:
```
x_clean = [1 2 4 5 6 7 8 9 10 11 13 14 15]
y_clean = [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]';
```

Least Squares

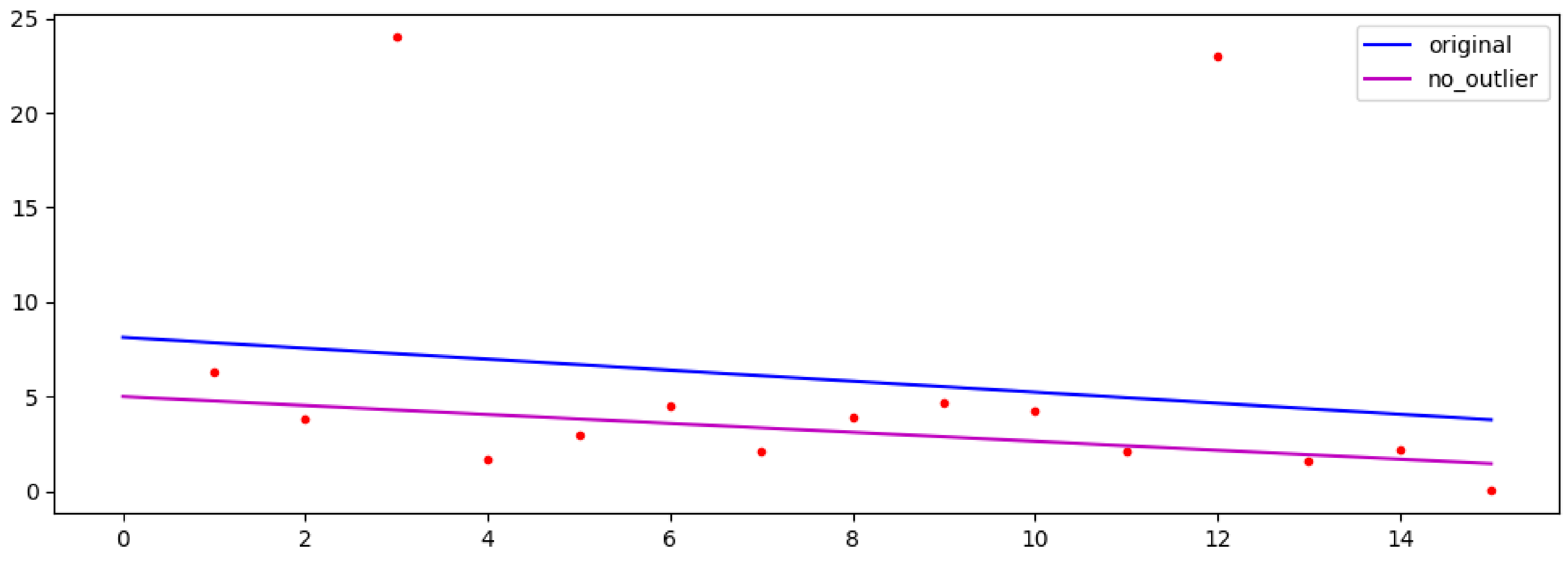
In [7]:
```
# order of polynomial to use
k=1
# fit using a function of the form f(x) = u1 x^k + u2 x^(k-1) + ... + uk x + u{k+1}
n_clean = length(x_clean)
A_clean = zeros(n_clean,k+1)
for i = 1:n_clean
    for j = 1:k+1
        A_clean[i,j] = x_clean[i]^(k+1-j)
    end
end
```

In [8]:
```
m = Model(solver=GurobiSolver(OutputFlag=0))
@variable(m, u[1:k+1])
@objective(m, Min, sum((y_clean - A_clean*u).^2))

status = solve(m)
uopt_clean = getvalue(u)
println(status)
println("L2 Error: ",norm(y_clean - A_clean*uopt_clean),
    " Objective value: ",getobjectivevalue(m),", Parameters learnt: ", uopt_clean)
```

```
Optimal
L2 Error: 4.528647342002823 Objective value: 20.508646748243223, Parameters learnt: [-0.236484,4.991
6]
```

Plot

In [9]:
```
using PyPlot
npts = 100
xfine = linspace(0,15,npts)
ffine = ones(npts)
for j = 1:k
    ffine = [ffine.*xfine ones(npts)]
end
yfine = ffine * uopt

xfine_clean = linspace(0,15,npts)
ffine_clean = ones(npts)
for j = 1:k
    ffine_clean = [ffine_clean.*xfine_clean ones(npts)]
end
yfine_clean = ffine_clean * uopt_clean

figure(figsize=(12,4))
plot( x, y, "r.")
plot( xfine, yfine, "b-",label="original");
plot( xfine_clean, yfine_clean, "m-", label="no_outlier")
legend();
```



The difference between the two lines can be explained by the fact that the model for the clean data doesn't try to minmize the cost which it incurs due to the outliers which is pretty significant if it has to approximate the other 13 points.

b) 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 ℓ_1 cost function:

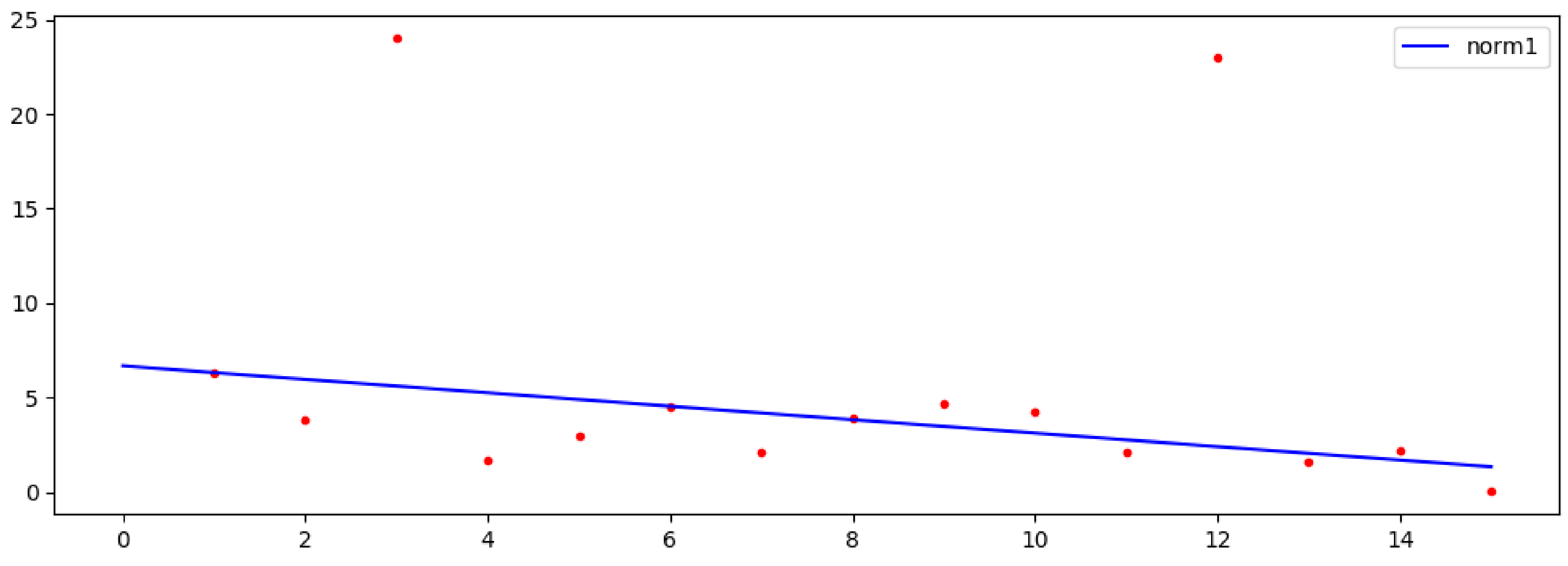
$$\ell_1\ cost: \sum_{i=1}^{15} |y_i - ax_i - b|$$

Include a plot containing the data and the best ℓ_1 linear fit. Does the ℓ_1 cost handle outliers better or worse than least squares? Explain why.

In [10]:
```
m = Model(solver=GurobiSolver(OutputFlag=0))
@variable(m, uopt_norm1[1:k+1])
@objective(m, Min, sum(abs_array(y - A*uopt_norm1)))

status = solve(m)
uopt_L1 = getvalue(uopt_norm1)
println(status)
println("L2 Error: ",norm(y - A*uopt_L1)," Objective Value: ",getobjectivevalue(m),
    ", Parameters learnt: ", uopt_L1)
```

```
Optimal
L2 Error: 28.17265589183952 Objective Value: 54.029999999999994, Parameters learnt: [-0.356,6.666]
```

In [11]:
```
xfine_L1 = linspace(0,15,npts)
ffine_L1 = ones(npts)
for j = 1:k
    ffine_L1 = [ffine_L1.*xfine_L1 ones(npts)]
end
yfine_L1 = ffine_L1 * uopt_L1

figure(figsize=(12,4))
plot( x, y, "r.")
plot( xfine_L1, yfine_L1, "b-",label="norm1")
legend();
```



The ℓ_1 norm model fits the data much better. The ℓ_2 norm penalizes the model much severly for the outliers since the distance to them is squared, whereas the ℓ_1 norm is just adding the absolute values. This results in less shifting of the line towards the outlier in the ℓ_1 norm case and hence the better results

c) Another approach is to use an ℓ_2 penalty for points that are close to the line but an ℓ_1 penalty for points that are far away. Specifically, we'll use something called the Huber loss, defined as:

$$\phi(x) = \begin{cases} x^2 & if -M \le x \le M \\ 2M|x| - M^2 & otherwise \end{cases}$$

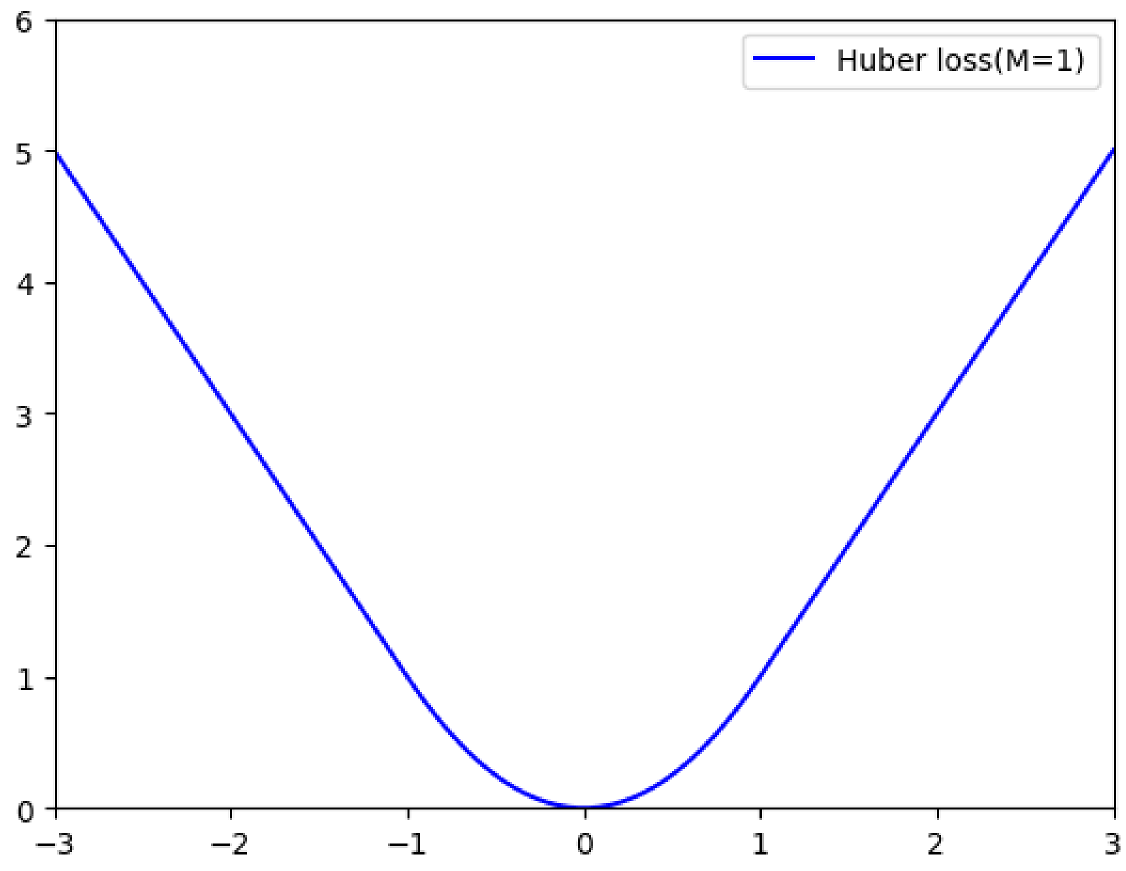
Here, M is a parameter that determines where the quadratic function transitions to a linear function. The plot on the right shows what the Huber loss function looks like for M = 1. The formula above is simple, but not in a form that is useful for us. As it turns out, we can evaluate the Huber loss function at any point x by solving the following convex QP instead:

$$\phi(x) = \begin{cases} \underset{v,w}{\text{minimize}} & w^2 + 2Mv \\ \text{subject to:} & |x| \le w + v \\ & v \ge 0, w \le M. \end{cases}$$

Verify this fact by solving the above QP (with M = 1) for many values of x in the interval $-3 \le x \le 3$ and reproducing the plot above. Finally, find the best linear fit to our data using a Huber loss with $M = 1$ and produce a plot showing your fit. The cost function is:

$$Huber\ Loss : \sum_{i=1}^{15} \phi(y_i - ax_i - b)$$

In [12]:
```
M = 1
X = linspace(-3,3,npts)
Y = zeros(1,100)
for (i,pt) in  enumerate(X)
    m = Model(solver=GurobiSolver(OutputFlag=0))

    @variable(m, v >= 0)
    @variable(m, w <= M)
    @constraint(m, abs(pt) <= w + v)
    @objective(m, Min, w^2 + 2M*v)
    solve(m)
    Y[i] = getobjectivevalue(m)
end

plot(X, Y', "b-", label="Huber loss(M=1)")
xlim(-3,3)
ylim(0,6)
legend();
```

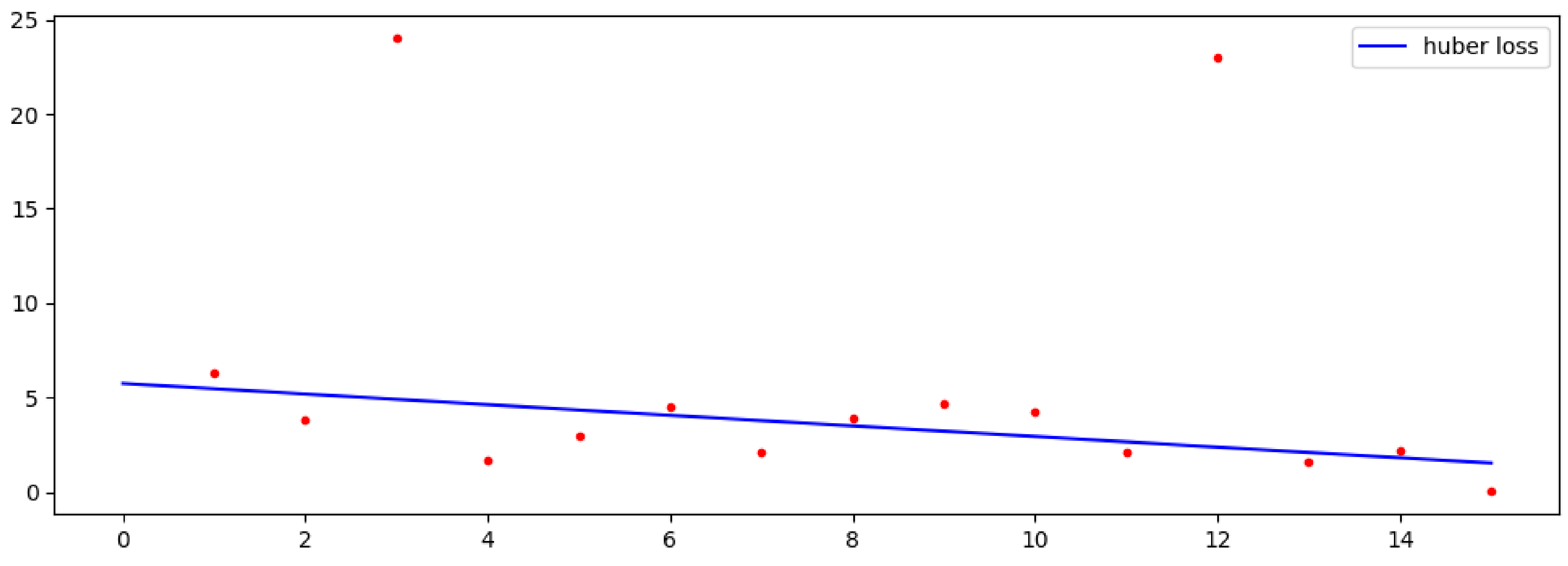


The plot for different values from interval $[-3, 3]$ is as given in the question

In [13]:
```
m = Model(solver=GurobiSolver(OutputFlag=0))

@variable(m, uopt_huber[1:k+1])
@variable(m, v[1:15] >= 0)
@variable(m, w[1:15] <= M)

@constraint(m, abs_array(y - A*uopt_huber) .<= w + v)

@objective(m, Min, sum(w.^2 + 2M*v))
status = solve(m)
uopt_Huber = getvalue(uopt_huber)
println(status)
println("L2 Error: ", norm(y-A*uopt_Huber)," Objective Value: ",
    getobjectivevalue(m),", Parameters learnt: ", uopt_Huber)
```

```
Optimal
L2 Error: 28.526155756854973 Objective Value: 95.49973754752351, Parameters learnt: [-0.281108,5.738
12]
```

In [14]:
```
xfine_huber = linspace(0,15,npts)
ffine_huber = ones(npts)
for j = 1:k
    ffine_huber = [ffine_huber.*xfine_huber ones(npts)]
end
yfine_huber = ffine_huber * uopt_Huber

figure(figsize=(12,4))
plot( x, y, "r.")
plot( xfine_huber, yfine_huber, "b-",label="huber loss")
legend();
```



Homework 6 Question 2: Hyperbolic program

In this problem, we start with a problem that doesn't appear to be convex and show that it is in fact convex by converting it into an SOCP.

a) Recall from class that for any w ∈ R^n and x_o, y ∈ R, the following constraints are equivalent:

$$w^T w \leq x_o y, x_o \geq 0, y \geq 0 \iff || \begin{bmatrix} 2w \\ x_o - y \end{bmatrix} || \leq x_o + y$$

Suppose we have an optimization problem with variables t ≥ 0 and x ∈ R^n. Express the constraint: $t(a^T x + b_o) \geq 1$ as a second-order cone constraint. Specifically, write the constraint in the form $||Ax + b|| \leq c^T x + d$. What are A, b, c, d, x?

Equation $t(a^T x + b_o) \geq 1$ can be represented as $w^T w \leq x_o y$ by using following substitutions.

$$x_o = a^T x + b_o$$
$$y = t$$

$$w = [1]$$

a 1x1 matrix such that

$$w^T w = 1$$

Now the same equation $w^T w \leq x_o y$ can be written as an SOCP

$$|| \begin{bmatrix} 2w \\ x_o - y \end{bmatrix} || \leq x_o + y$$

Which is equivalent to

$$|| \begin{bmatrix} 2[1] \\ a^T x + b_o - t \end{bmatrix} || \leq a^T x + b_o + t$$

To convert the above equation to the standard form of

$$||Ax + b|| \leq c^T x + d$$

the following substitutions are done based on the dimension of the matrices

$$|| \begin{bmatrix} 0 \\ a^T \end{bmatrix} x + \begin{bmatrix} 2[1] \\ b_o - t \end{bmatrix} || \leq a^T x + b_o + t$$

Finally to answer the question,

$$A = \begin{bmatrix} 0 \\ a^T \end{bmatrix}$$

$$b = \begin{bmatrix} 2[1] \\ b_o - t \end{bmatrix}$$

$$c = a$$

$$d = b_o + t$$

$$x = x$$

A is a 2xn matrix, where first row is zero. b is a 2x1 vector. c is of the same dimension as of a.

Note: I could have taken w as any normal vector ∈ R^n, then there will be aditional rows added to Matrix A. As there were multiple solutions possible to this question through the choice of w, I made the simplest one.

b) Consider the following hyperbolic optimization problem (note the nonlinear objective):

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{i=1}^{p} 1/(a_i^T x + b_i) \\ \text{subject to:} \quad & a_i^T x + b_i > 0, i = 1, \ldots, p \\ & c_j^T x + d_j \geq 0, j = 1, \ldots, q. \end{aligned}$$

Write this optimization problem as an SOCP.

For all the expressions

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

I take an auxillary variable $t_i \geq 0, i = 1, \ldots, p$ and modify the given constraint as

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

Also the objective will now chage to

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

. This addition doesn't scale the objective function in any way but just shifts the function by 1. The optimal value of x can be calculated by solving this function and just shifting it back.

Now the optimization model can have the following constraint

$$t_i(a_i^T x + b_i + 1) \geq 1, i = 1, \ldots, p$$

This constraint is equivalent to the one solve in part a). Using the constraint just added and the new objective function the given optimization problem is converted to the following which uses SOCP.

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \sum_{i=1}^{p} t_i \\ \text{subject to:} & \| \begin{bmatrix} 0 \\ a_i^T \end{bmatrix} x + \begin{bmatrix} 2[1] \\ b_i + 1 - t_i \end{bmatrix} \| \leq a_i^T x + b_i + 1 + t_i, i = 1, \ldots, p \\ & a_i^T x + b_i > 0, i = 1, \ldots, p \\ & c_j^T x + d_j \geq 0, j = 1, \ldots, q. \end{array}$$

since

$$t_i \geq 1/(a_i^T x + b_i + 1)$$

so minimizing t and the expression on the RHS is equivalent.

The above problem is a standard SOCP problem, since the objective is a linear function and atleast one of the constraints is SOCP.

Homework 6 Question 3: Heat pipe design.

A heated fluid at temperature T (degrees above ambient temperature) flows in a pipe with fixed length and circular cross section with radius r. A layer of insulation, with thickness w, surrounds the pipe to reduce heat loss through the pipe walls (w is much smaller than r). The design variables in this problem are T, r, and w.

The energy cost due to heat loss is roughly equal to $\alpha_1 Tr/w$. The cost of the pipe, which has a fixed wall thickness, is approximately proportional to the total material, i.e., it is given by $\alpha_2 r$. The cost of the insulation is also approximately proportional to the total insulation material, i.e., roughly $\alpha_3 rw$. The total cost is the sum of these three costs.

The heat flow down the pipe is entirely due to the flow of the fluid, which has a fixed velocity, i.e., it is given by $\alpha_4 Tr^2$. The constants α_i are all positive, as are the variables T, r, and w.

Now the problem: maximize the total heat flow down the pipe, subject to an upper limit C_max on total cost, and the constraints

$$T_{min} \le T \le T_{max},\ r_{min} \le r \le r_{max},\ w_{min} \le w \le w_{max},\ w \le 0.1r$$

a) Express this problem as a geometric program, and convert it into a convex optimization problem.

Equivalent G.P.:

$$\begin{array}{ll} \text{minimize} & (1/\alpha_4)T^{-1}r^{-2} \\ \text{subject to:} & (1/T_{max})T \le 1, T_{min}T^{-1} \le 1 \\ & (1/r_{max})r \le 1, r_{min}r^{-1} \le 1 \\ & (1/w_{max})w \le 1, w_{min}w^{-1} \le 1 \\ & 10wr^{-1} \le 1. \end{array}$$

Let $x = log(T), y = log(r), z = log(w)$

So objective function

$$\alpha_4 Tr^2 : log(e^{log(\alpha_4)+x+2y})$$

Constraints

$$\begin{array}{c} T_{min} \le T \le T_{max} :\ log(e^{log(T_{min})-x}) \le 0,\ log(e^{x+log(1/T_{max})}) \le 0 \\ r_{min} \le r \le r_{max} :\ log(e^{log(r_{min})-y}) \le 0,\ log(e^{y+log(1/r_{max})}) \le 0 \\ w_{min} \le w \le w_{max} :\ log(e^{log(w_{min})-z}) \le 0,\ log(e^{z+log(1/w_{max})}) \le 0 \\ w \le 0.1r : log(e^{log(10)+z-y}) \le 0 \\ ((\alpha_1/C_{max})Trw^{-1}) + ((\alpha_2/C_{max})r) + ((\alpha_3/C_{max})rw) <= 1 \end{array}$$

Heat Loss Cost $((\alpha_1/C_{max})Trw^{-1}) => log(e^{log(\alpha_1/C_{max})+x+y-z})$

Pipe Cost $((\alpha_2/C_{max})r) => log(e^{log(\alpha_2/C_{max})+y})$

Insulation Cost $((\alpha_3/C_{max})rw) => log(e^{log(\alpha_3/C_{max})+y+z})$

b) Consider a simple instance of this problem, where $Cmax = 500$ and $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 1$. 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 [1]:
```
using JuMP, Mosek

α1 = 1
α2 = 1
α3 = 1
α4 = 1
Cmax = 500

m = Model(solver=MosekSolver(LOG=0))

@variable(m, x)
@variable(m, y)
@variable(m, z)

@NLexpression(m, heatLossCost, e^(log(α1) - log(Cmax) + x + y - z))
@NLexpression(m, pipeCost, e^(log(α2) - log(Cmax) + y))
@NLexpression(m, insulationCost, e^(log(α3) - log(Cmax) + y + z))

@NLconstraint(m, log(10) + z - y  <= 0)
@NLconstraint(m, log(heatLossCost + pipeCost + insulationCost) <= 0)

@NLexpression(m, heatFlow, -log(α4) - x - 2y)

@NLobjective(m, Min, heatFlow)
println("Status:", solve(m))
X = getvalue(x)
Y = getvalue(y)
Z = getvalue(z)

T = e^X
r = e^Y
w = e^Z
println("Optimal Temperature: ", T, ", Radius: ",r, ", Width: ",w)
println("Total cost: ",α1*(T*r)/w + α2*r + α3*r*w)
println("Max heat flow: ", α4*T*r^2)
```

```
Status:Optimal
Optimal Temperature: 23.840213386281093, Radius: 46.39045139252367, Width: 4.639040234094068
Total cost: 500.0000078210848
Max heat flow: 51305.90291584941
```