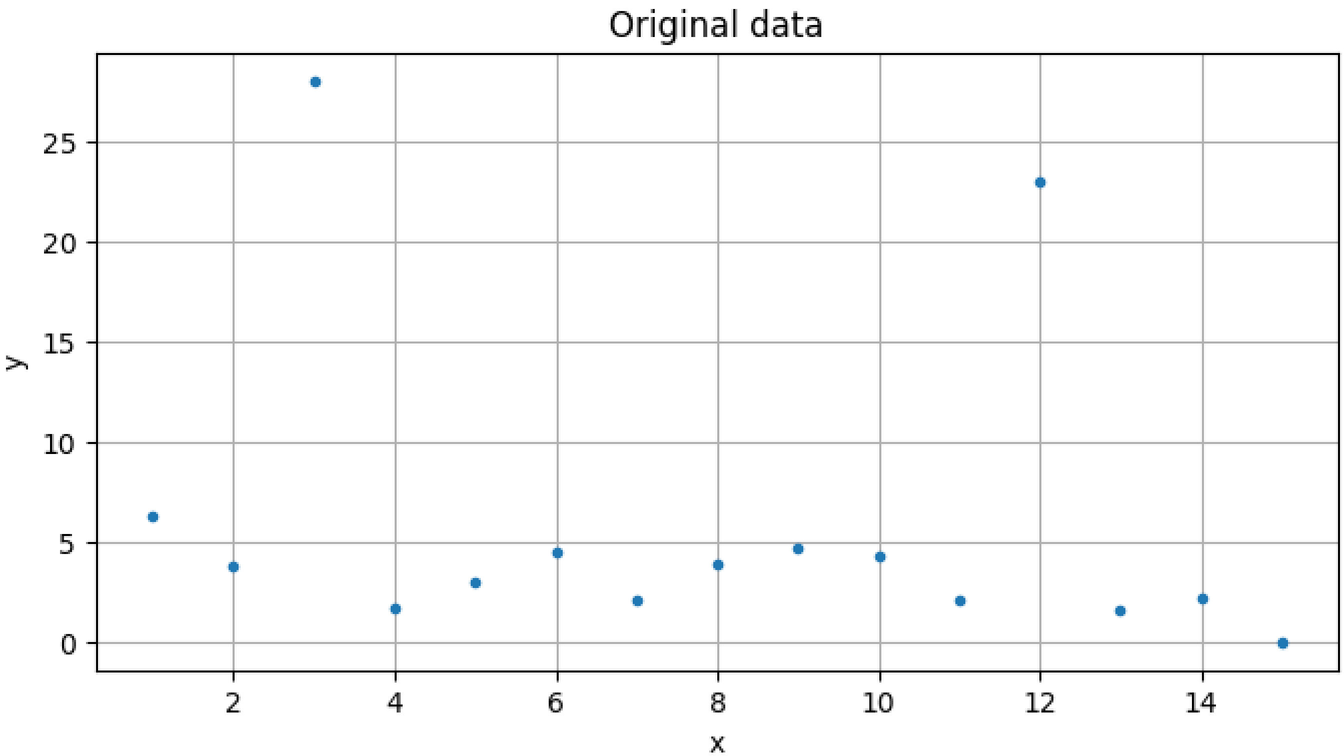
1. The Huber Loss [15 pts]

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	28.0	1.71	2.99	4.53	2.11	3.88	4.67	4.25	2.06	23.0	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.

In [1]:
```
x = [1:15;]
y = [6.31, 3.78, 28.0, 1.71, 2.99, 4.53, 2.11, 3.88, 4.67, 4.25, 2.06, 23.0, 1.58, 2.17, 0.02]

using PyPlot

figure(figsize=(8,4))
title("Original data")
plot(x, y, ".")
xlabel("x")
ylabel("y")
grid("on")
```



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:

$$\ell_2 \text{ cost:} \qquad \sum_{i=1}^{15} (y_i - a x_i - b)^2$$

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 [2]:

```
# without removing outliers, using L_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

using JuMP, Gurobi, Mosek

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

@variable(m, u[1:k+1])
@objective(m, Min, sum( (y - A*u).^2 ) )
status = solve(m)
uopt = getvalue(u)
println(status)
println(uopt)
```

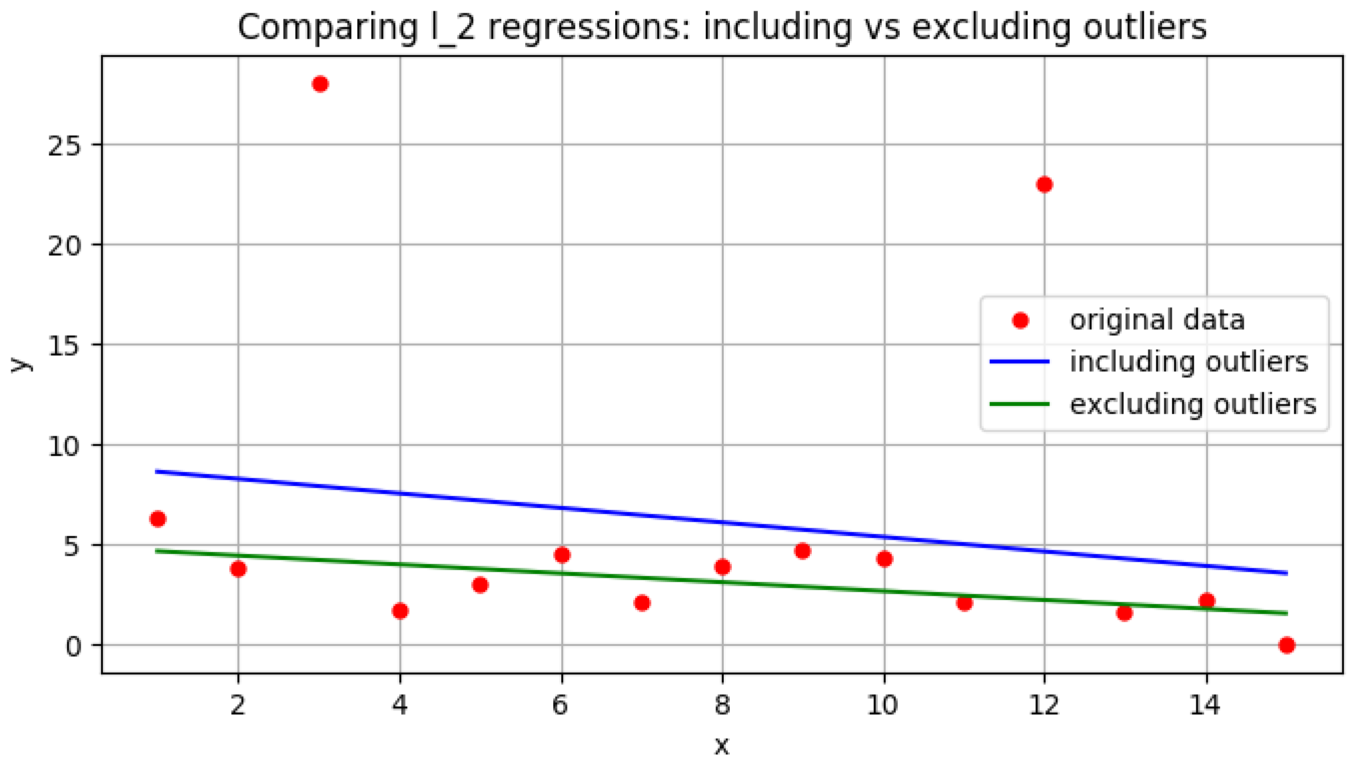
```
Optimal
[-0.362214,8.96838]
```

In [3]:
```julia
# with removed outliers using L_2
x1 = [1:13;]
y1 = [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]

# 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}
n1 = length(x1)
A1 = zeros(n1,k+1)
for i = 1:n1
    for j = 1:k+1
        A1[i,j] = x1[i]^(k+1-j)
    end
end

using JuMP, Gurobi, Mosek

m1 = Model(solver=MosekSolver(LOG=0))

@variable(m1, u1[1:k+1])
@objective(m1, Min, sum( (y1 - A1*u1).^2 ) )
status1 = solve(m1)
uopt1 = getvalue(u1)
println(status1)
println(uopt1)
```

```
Optimal
[-0.258791,4.89308]
```

In [4]:

```
using PyPlot

npts = 100
xfine = linspace(x[1], x[end], npts)
xfine1 = linspace(x1[1], x1[end], npts)
ffine = ones(npts)
ffine1 = ones(npts)

for j = 1:k
    ffine = [ffine.*xfine ones(npts)]
    ffine1 = [ffine1.*xfine1 ones(npts)]
end
yfine = ffine * uopt
yfine1 = ffine1 * uopt1

figure(figsize=(8,4))
title("Comparing l_2 regressions: including vs excluding outliers")
plot( x, y, "r.", markersize=10)
plot( xfine, yfine, "b-")
plot( xfine, yfine1, "g-")
legend(["original data", "including outliers", "excluding outliers"], loc="right")
xlabel("x")
ylabel("y")
grid()
;
```



The outliers included in the first regression line essentially shifted the true regression line upwards. We notice a slight change of the gradient too, but for this data set, the effect could be considered negligible.

In the second regression line, we see that the line touches some of the plotted data, making it a btter line of best fit, because it minimizes the error in y at each x

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:

$$\ell_1 \text{ cost:} \qquad \sum_{i=1}^{15} |\, y_i - a x_i - b \,|$$

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 [5]:

```julia
using JuMP, Gurobi

import JuMP: GenericAffExpr

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;
```

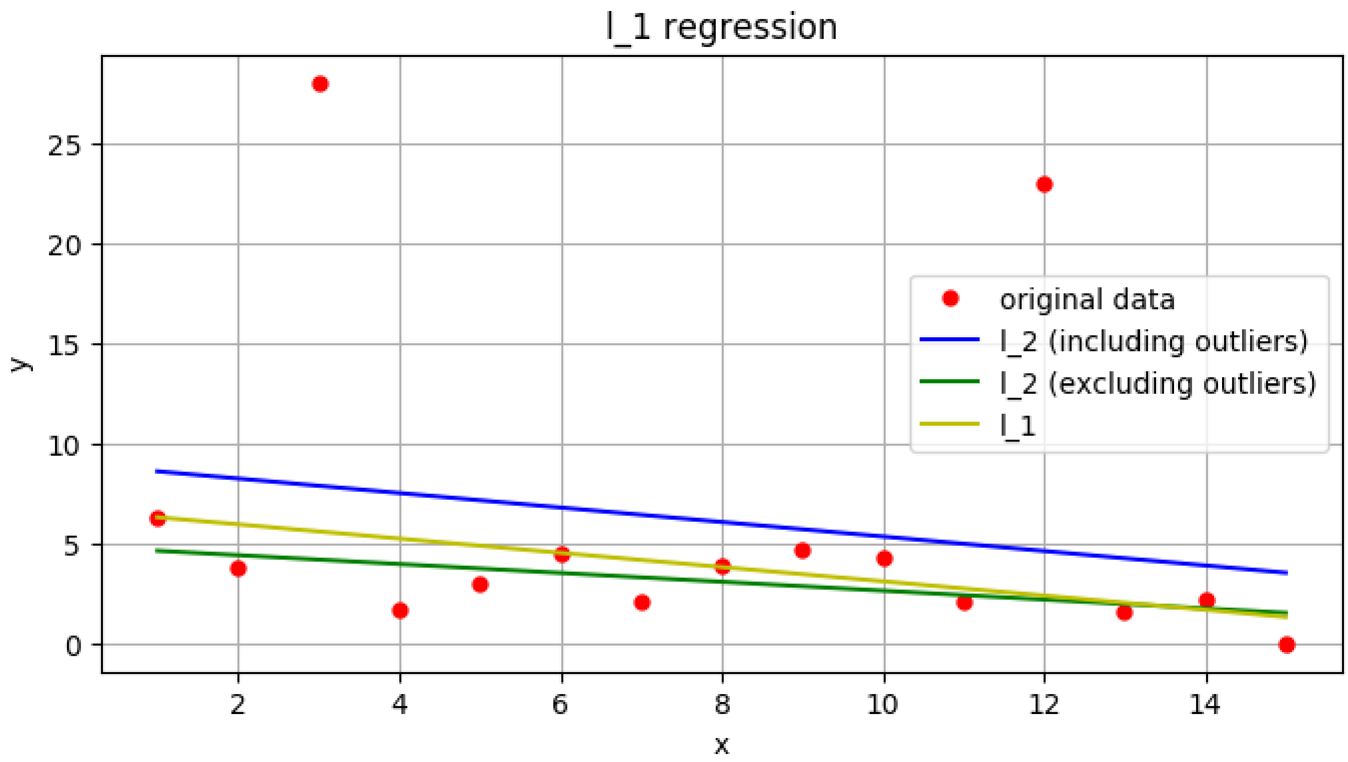
In [6]:

```julia
# 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}
n2 = length(x)
A2 = zeros(n2,k+1)
for i = 1:n2
    for j = 1:k+1
        A2[i,j] = x[i]^(k+1-j)
    end
end

using JuMP, Gurobi, Mosek

m2 = Model(solver=MosekSolver(LOG=0))
@variable(m2, u2[1:k+1])
#@variable(m2, t)
#@constraint(m2, -t <= y - A2*u2 <= t)
#@objective(m, Min, t)

@objective(m2, Min, sum( abs_array(y - A2*u2) ) )

status2 = solve(m2)
uopt2 = getvalue(u2)
println(status)
println(uopt2)
```

```
Optimal
[-0.356,6.666]
```

In [7]:

```
using PyPlot

npts = 100
xfine2 = linspace(x[1], x[end], npts)
ffine2 = ones(npts)

for j = 1:k
    ffine2 = [ffine2.*xfine2 ones(npts)]
end
yfine2 = ffine2 * uopt2

figure(figsize=(8,4))
title("l_1 regression")
plot( x, y, "r.", markersize=10)
plot( xfine, yfine, "b-")
plot( xfine, yfine1, "g-")
plot( xfine2, yfine2, "y-")
legend(["original data", "l_2 (including outliers)", "l_2 (excluding outliers)", "l_1"], loc="right")
xlabel("x")
ylabel("y")
grid()
;
```



In [8]:
```
println("error in m1 (l_2): ", getobjectivevalue(m1))
println("error in m2 (l_1): ", getobjectivevalue(m2))
```

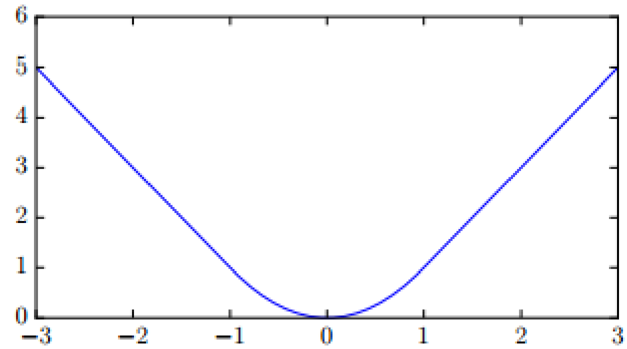
```
error in m1 (l_2): 21.681303296703305
error in m2 (l_1): 58.03000000020826
```

l_2 handles the line of best fit better, simply because the data isn't distorted *at all* by the outliers, whereas l_1 still faces some of the distortion due to them.

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:

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

Here, M is a parameter that determines where the quadratic function transitions to a linear function. The plot below 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:

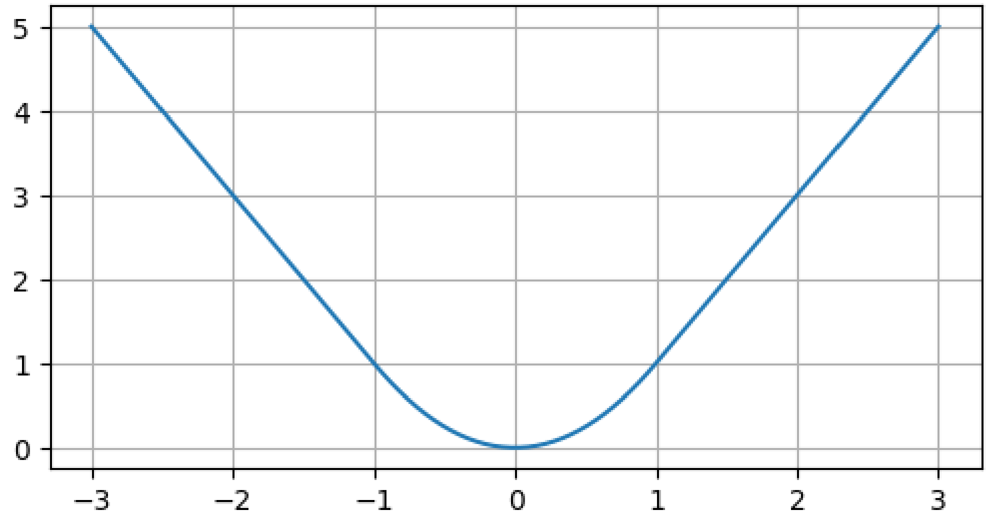
$$\phi(x) = \left\{ \begin{array}{ll} \underset{v,w}{\text{minimize}} & w^2 + 2Mv \\ \text{subject to:} & |x| \le w + v \\ & v \ge 0, \;\; w \le M \end{array} \right\}$$

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:

$$\text{Huber loss:} \qquad \sum_{i=1}^{15} \phi(\, y_i - a x_i - b)$$

In [9]:
```
using JuMP, PyPlot, Mosek

function getY(x, M)

    m3 = Model(solver=MosekSolver(LOG=0))

    @variable(m3, v >= 0)
    @variable(m3, w <= M)

    @constraint(m3, abs(x) <= w + v)

    @objective(m3, Min, w^2 + 2*M*v)

    solve(m3)

    return getobjectivevalue(m3)
end;
```

In [10]:
```
M = 1
x_h = linspace(-3, 3, 100)

y_h = zeros(length(x_h))
for i = 1:length(x_h)
    y_h[i] = getY(x_h[i], M)
end

figure(figsize=(6,3))
plot(x_h,y_h)
grid();
```

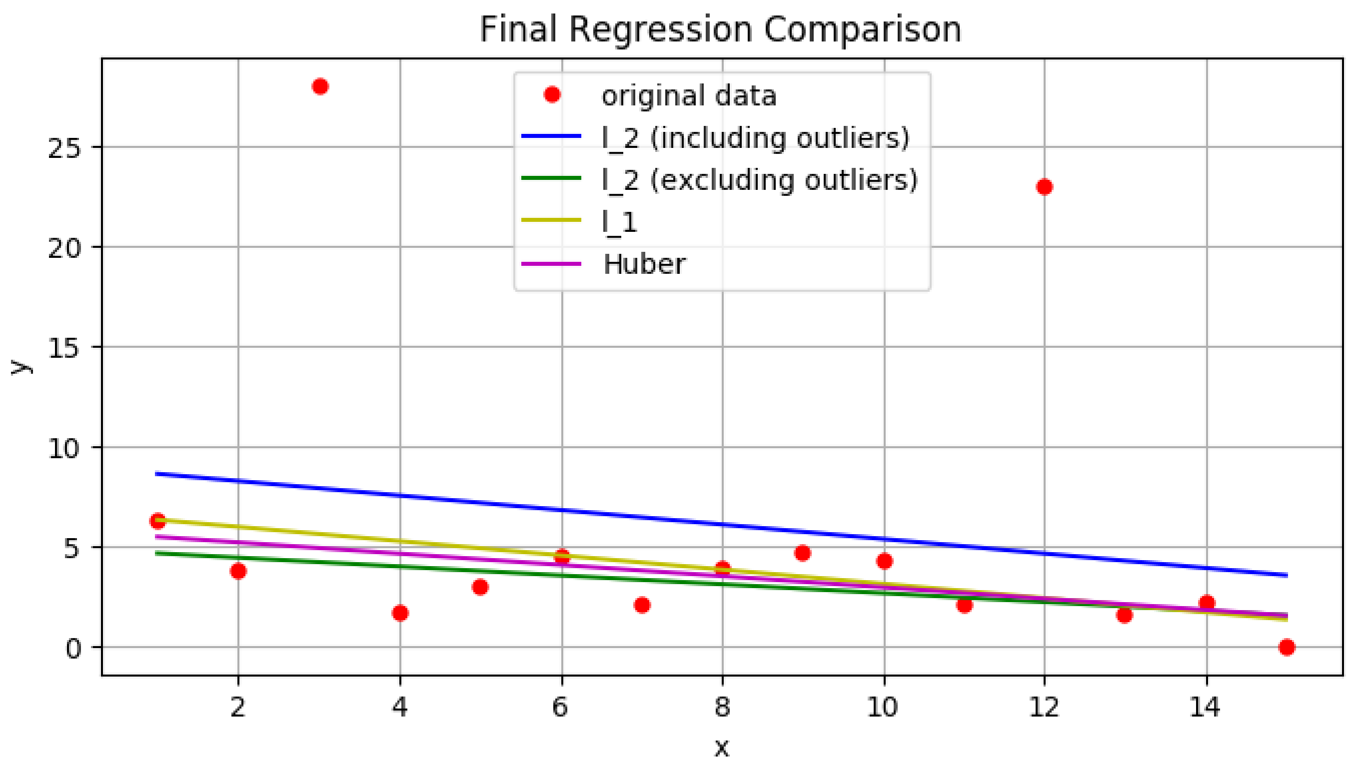


In [11]:

```julia
# 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}
n4 = length(x)
A4 = zeros(n4,k+1)
for i = 1:n4
    for j = 1:k+1
        A4[i,j] = x[i]^(k+1-j)
    end
end

using JuMP, Gurobi, Mosek

m4 = Model(solver=MosekSolver(LOG=0))

M = 1
@variable(m4, u4[1:k+1])
@variable(m4, v[1:n4] >= 0)
@variable(m4, w[1:n4] <= M)

@constraint(m4, abs_array(y - A4*u4) .<= w + v)

@objective(m4, Min, sum(w[i]^2 + 2*M*v[i] for i in 1:n4))

status4 = solve(m4)
uopt4 = getvalue(u4)
println(status4)
println(uopt4)
```

```
Optimal
[-0.281108,5.73812]
```

In [13]:
```
figure(figsize=(8,4))
title("Final Regression Comparison")
plot( x, y, "r.", markersize=10)
plot( xfine, yfine, "b-")
plot( xfine, yfine1, "g-")
plot( xfine2, yfine2, "y-")
plot( x, A*uopt4, "m-")
legend(["original data", "l_2 (including outliers)", "l_2 (excluding outliers)", "l_1", "Huber"], loc="top")
xlabel("x")
ylabel("y")
grid()
```



2. Hyperbolic program [10 pts]

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 $w \in \mathbb{R}^n$ and $x, y \in \mathbb{R}$, the following constraints are equivalent:

$$w^\top w \le xy, \quad x \ge 0, \quad y \ge 0 \qquad \Longleftrightarrow \qquad \left\|\begin{bmatrix} 2w \\ x-y \end{bmatrix}\right\| \le x+y$$

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

2(a)

Note: $w^T w \le xy$, $x \ge 0$, $y \ge 0$ $\iff$ $\left\|\begin{bmatrix} 2w \\ x-y \end{bmatrix}\right\| \le x+y$

Problem: $t \ge 0$, $x \in \mathbb{R}^n$

Express $t(a^T x + b) \ge 1$ as an SOCC

i.e. $\|Ax + b\| \le c^T x + d$

$t(a^T x + b) \ge 1$

let $y = a^T x + b$ $\Rightarrow$ $\left\|\begin{bmatrix} 0 & 0 \\ a^T & -1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ b \end{bmatrix}\right\| \le \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\therefore$ $t(a^T x + b) \ge 1$

$\equiv$ $(a^T x + b) \le -1/t$ $\Rightarrow$ $\left\|\underbrace{\begin{bmatrix} 0 & 0 \\ a^T & -1 \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ b \end{bmatrix}}_{b}\right\| \le \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}_{c^T}\underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ -1/t \end{bmatrix}}_{d}$

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

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

Write this optimization problem as an SOCP. *Hint: the first part of this problem is **very** relevant!*

(b)

$$
\begin{array}{lll}
\min_{x} & \sum_{i=1}^{p} (a_i^{T} x + b_i)^{-1} & \\
\text{s.t.} & a_i^{T} x + b_i > 0, & \text{for } i = 1, \ldots, p \\
 & c_j^{T} x + d_j \geq 0, & \text{for } j = 1, \ldots, q
\end{array}
$$

SOLUTION:

$$
\min_{x, w_i} \sum_{i=1}^{p} (w_i)
$$

$$
0 < \frac{1}{a_i^{T} x + b_i} \leq w_i \quad \text{for } i = 1, \ldots, p \Rightarrow \underbrace{1 \leq w_i(a_i^{T} x + b_i)}_{\text{of the same form as part (a)}} \text{ for } i = 1, \ldots,
$$

$$
\begin{array}{ll}
a_i^{T} x + b_i > 0, & \text{for } i = 1, \ldots, p \\
c_j^{T} x + d_j \geq 0, & \text{for } j = 1, \ldots, q
\end{array}
$$

$$
x \geq 0
$$

3. Heat-Pipe Design [10 pts]

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 with $i \in \{1, 2, 3, 4\}$ 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

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

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

① $\max_{T,r,w \geq 0} \quad (\alpha_4 \cdot T \cdot r^2)$

$$\text{s.t.} \quad (\alpha_1 T \cdot r/w) + (\alpha_2 \cdot r) + (\alpha_3 \cdot r \cdot w) \leq c_{max}$$

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

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

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

$$w \leq 0.1 r$$

② $\min_{T,r,w \geq 0} \quad (\alpha_4^{-1} T^{-1} r^{-2})$

$$\text{s.t.} \quad \frac{T - T_{min}}{T_{max} - T_{min}} \leq 1 \qquad \frac{(\alpha_1 T r/w) + (\alpha_2 \cdot r) + (\alpha_3 \cdot r \cdot w)}{c_{max}} \leq 1$$

$$\frac{r - r_{min}}{r_{max} - r_{min}} \leq 1 \qquad 10\left(\frac{w}{r}\right) \leq 1$$

$$\frac{w - w_{min}}{w_{max} - w_{min}} \leq 1$$

③ Define $\quad x := \ln(T) \iff T = \exp(x)$

$$y := \ln(r) \iff r = \exp(y)$$

$$z := \ln(w) \iff w = \exp(z)$$

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

$$x \leq \ln(T_{max}) \qquad e^{z-y} \leq 0.1$$

$$y \leq \ln(r_{max})$$

$$z \leq \ln(w_{max})$$

$$\underbrace{\frac{e^{\ln(\alpha_1)+x+y-z} + e^{\ln(\alpha_2)+y} + e^{\ln(\alpha_3)+y+z}}{e^{\ln(c_{max})}} \leq 1}$$

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

$$\downarrow$$

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

b) Consider a simple instance of this problem, where $C_{max} = 500$ 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

a = ones(4)
C_max = 500

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

@variable(m, x >= 0)
@variable(m, y >= 0)
@variable(m, z >= 0)

@NLexpression(m, T, exp(x))
@NLexpression(m, r, exp(y))
@NLexpression(m, w, exp(z))

@NLexpression(m, heat_loss_cost, a[1]*T*r/w)
@NLexpression(m, pipe_cost, a[2]*r)
@NLexpression(m, insulation_cost, a[3]*r*w)
@NLexpression(m, total_cost, heat_loss_cost + pipe_cost + insulation_cost)

@NLconstraint(m, exp(log(a[1]) + x + y - z) + exp(log(a[2]) + y) + exp(log(a[3]) + y + z) - C_max <= 0)
@NLconstraint(m, exp(z - y) <= 0.1)

@NLobjective(m, Min, log(a[4]) - x - 2y)

println(solve(m))

println("We achieved a maximum heat flow of ", getvalue(T) * getvalue(r)^2)
println("T = ", getvalue(T))
println("r =  ", getvalue(r))
println("w = ", getvalue(w))
println("Our total cost is: ", getvalue(total_cost))
```

```
Optimal
We achieved a maximum heat flow of 51305.90644653668
T = 23.840238958644314
r =  46.39042810824172
w = 4.639042747423223
Our total cost is: 500.0000000182839
```

In [2]:

```julia
# attempting something I saw on Piazza (@196)
# inverted objective, then solve for minimization
# NOT MY FINAL ANSWER

using JuMP, Mosek

m1 = Model(solver=MosekSolver(LOG=0))

@variable(m1, T >= 0)
@variable(m1, r >= 0)
@variable(m1, w >= 0)

@NLexpression(m1, x, log(T))
@NLexpression(m1, y, log(r))
@NLexpression(m1, z, log(w))

@NLexpression(m1, heat_loss_cost, a[1]*T*r/w)
@NLexpression(m1, pipe_cost, a[2]*r)
@NLexpression(m1, insulation_cost, a[3]*r*w)
@NLexpression(m1, total_cost, heat_loss_cost + pipe_cost + insulation_cost)

@NLconstraint(m1, a[1]*T*r/w + a[2]*r + a[3]*r*w <= C_max)
@NLconstraint(m1, w <= 0.1*r)

@NLobjective(m1, Min, (a[4]*T*r^2)^-1)

println(solve(m1))

println("We achieved a maximum heat flow of ", getvalue(T) * getvalue(r)^2)
println("T = ", getvalue(T))
println("r =  ", getvalue(r))
println("w = ", getvalue(w))
println("Our total cost is: ", getvalue(total_cost))
```

```
Optimal
We achieved a maximum heat flow of 48667.67592771966
T = 20.24756171792293
r =  49.02684407505225
w = 4.3118627352713395
Our total cost is: 490.64317317489616
```