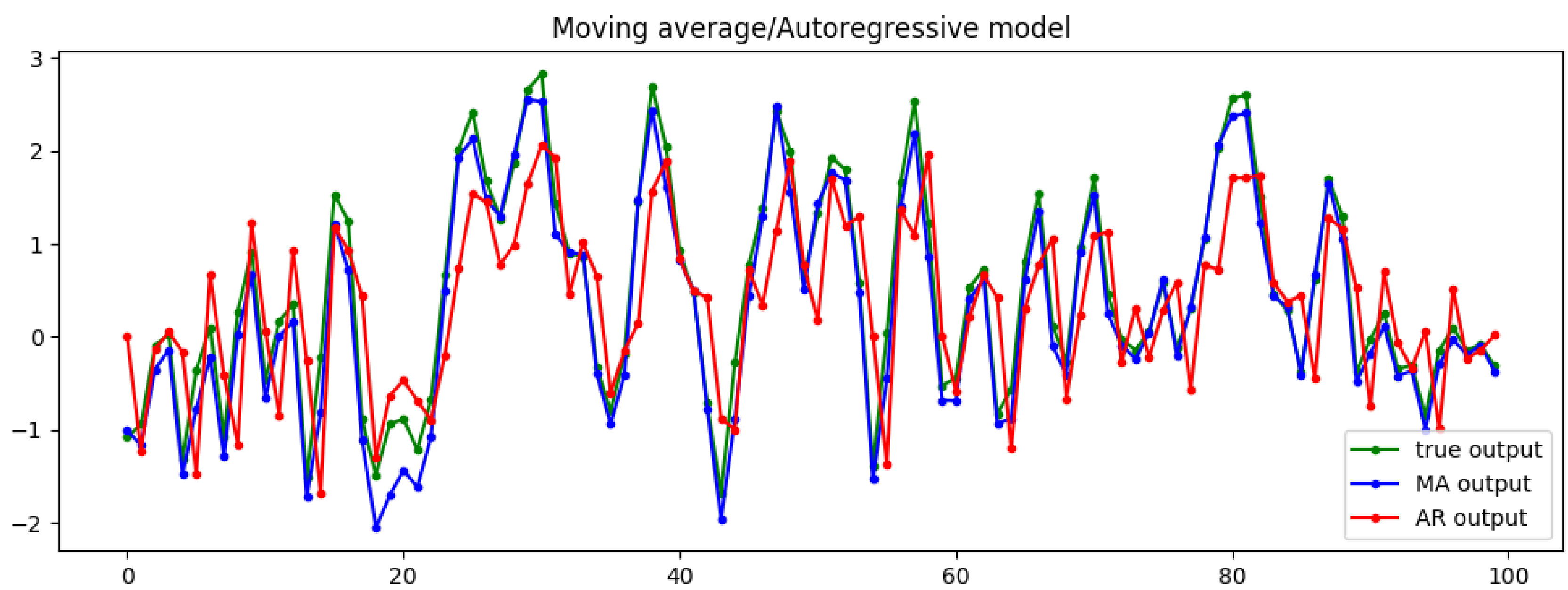
Moving Averages

Below we use both the moving average and autoregressive model and compare their estimations.

In [346]:
```
# Load the data file (ref: Boyd/263)
raw = readcsv("/home/john/Julia/uy_data.csv");
u = raw[:,1];
y = raw[:,2];
T = length(u);
```

In [347]:
```
# generate A matrix. Using more width creates better fit.  (MA model)
width = 5
A = zeros(T,width)
B = zeros(T,width)
for i = 1:width
    A[i:end,i] = u[1:end-i+1]
    B[(i+1):end,i] = y[1:end-i]
end

zopt = B\y
zest = B*zopt

wopt = A\y
yest = A*wopt

figure(figsize=(12,4))
plot(y,"g.-",yest,"b.-",zest,"r.-")
legend(["true output", "MA output", "AR output"], loc="lower right");
title("Moving average/Autoregressive model");
println("MA error: ", norm(yest-y))
println("AR error: ", norm(zest-y))
```

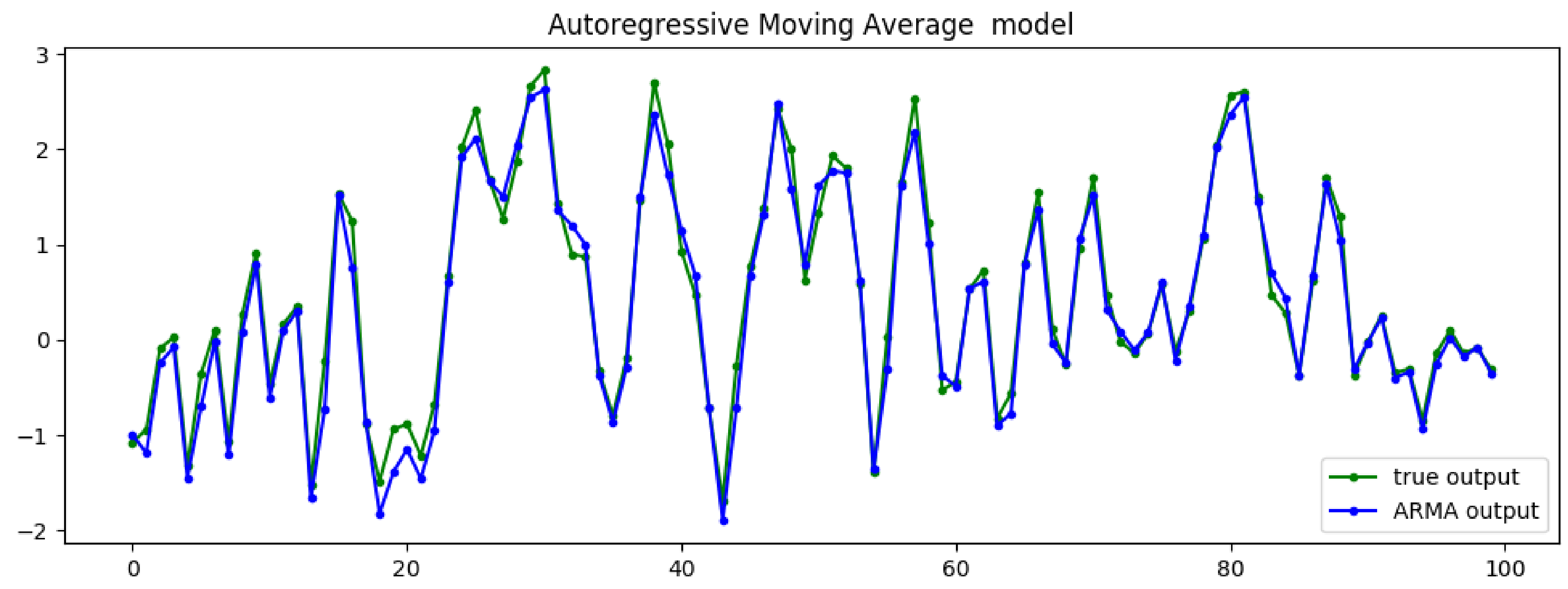



```
MA error: 2.460854388269911
AR error: 7.436691765656793
```

Here we combine moving averages and autoregression into one estimation method and calculate it's error.

In [348]:
```
width = 2
C = zeros(T,width)
for i = 2:T
    C[i,1] = y[i-1]
    C[i-1,2] = u[i-1]
end
C[T,2] = u[T]

mropt = C\y
mrest = C*mropt;
```

In [349]:
```
figure(figsize=(12,4))
plot(y,"g.-",mrest,"b.-")
legend(["true output", "ARMA output"], loc="lower right");
title("Autoregressive Moving Average  model");
println()
println("ARMA error: ", norm(mrest-y))
```




```
ARMA error: 1.8565828148734604
```

Voltage Smoothing

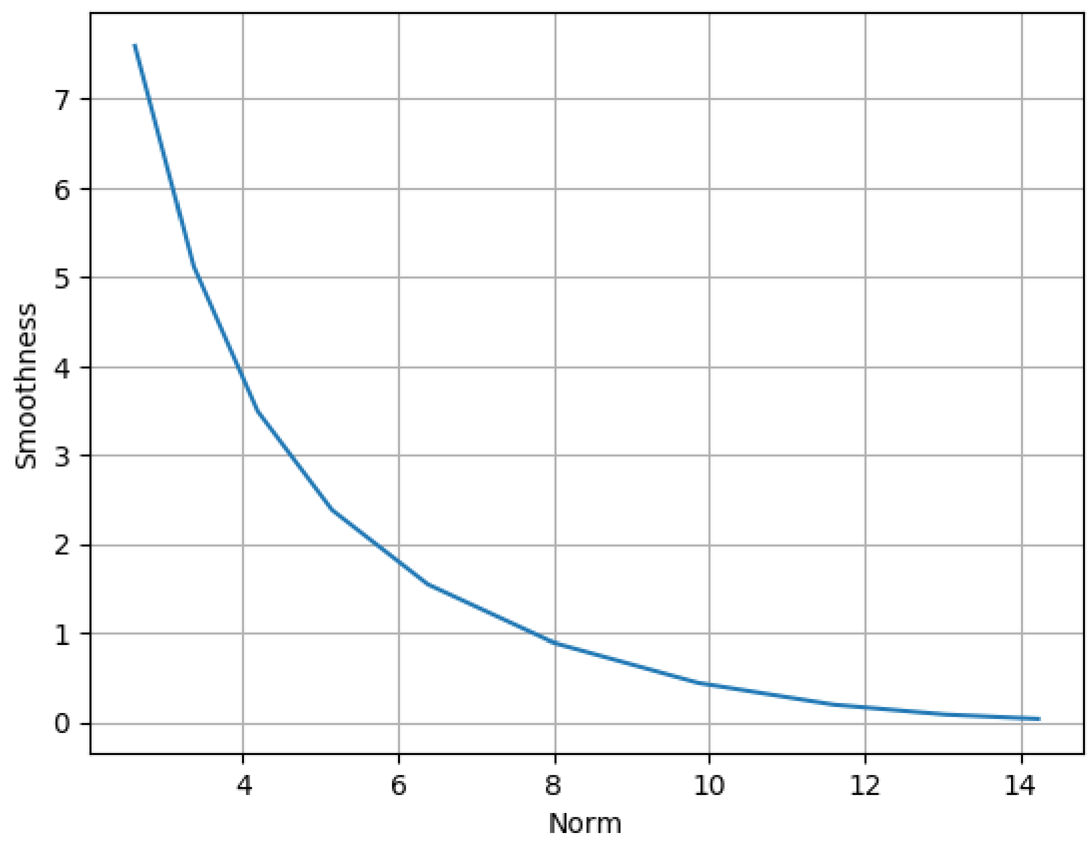
In [350]:
```
using JuMP
using Gurobi

raw = readcsv("/home/john/Julia/voltages.csv");
T = length(raw);
```

In [351]:
```
v = raw[:,1]

n = 10
lambda = logspace((1/10), 3, n)
J1 = zeros(n)
J2 = zeros(n)

for j = 1:n
   m = Model(solver=GurobiSolver())
   @variable(m, x[1:T])
   @objective(m, Min, sum((x - v).^2) + lambda[j]*(sum((x[i-1] - x[i])^2 for i = 2:T)))
   solve(m)
   y = getvalue(x)
   z = getobjectivevalue(m)
   J1[j] = norm(y - v)
   J2[j] = sum((y[i-1] - y[i])^2 for i = 2:T)
end
```

```
Optimize a model with 0 rows, 200 columns and 0 nonzeros
Model has 399 quadratic objective terms
Coefficient statistics:
  Matrix range     [0e+00, 0e+00]
  Objective range  [1e+00, 4e+00]
  QObjective range [5e+00, 7e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [0e+00, 0e+00]
Presolve time: 0.00s
Presolved: 0 rows, 200 columns, 0 nonzeros
Presolved model has 399 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 399
 AA' NZ     : 4.700e+02
 Factor NZ  : 2.449e+03
 Factor Ops : 3.695e+04 (less than 1 second per iteration)
 Threads    : 1
```

Tradeoff Graph

In [335]:
```
plot(J1, J2)
xlabel("Norm")
ylabel("Smoothness")
grid()
```



Spline Fitting

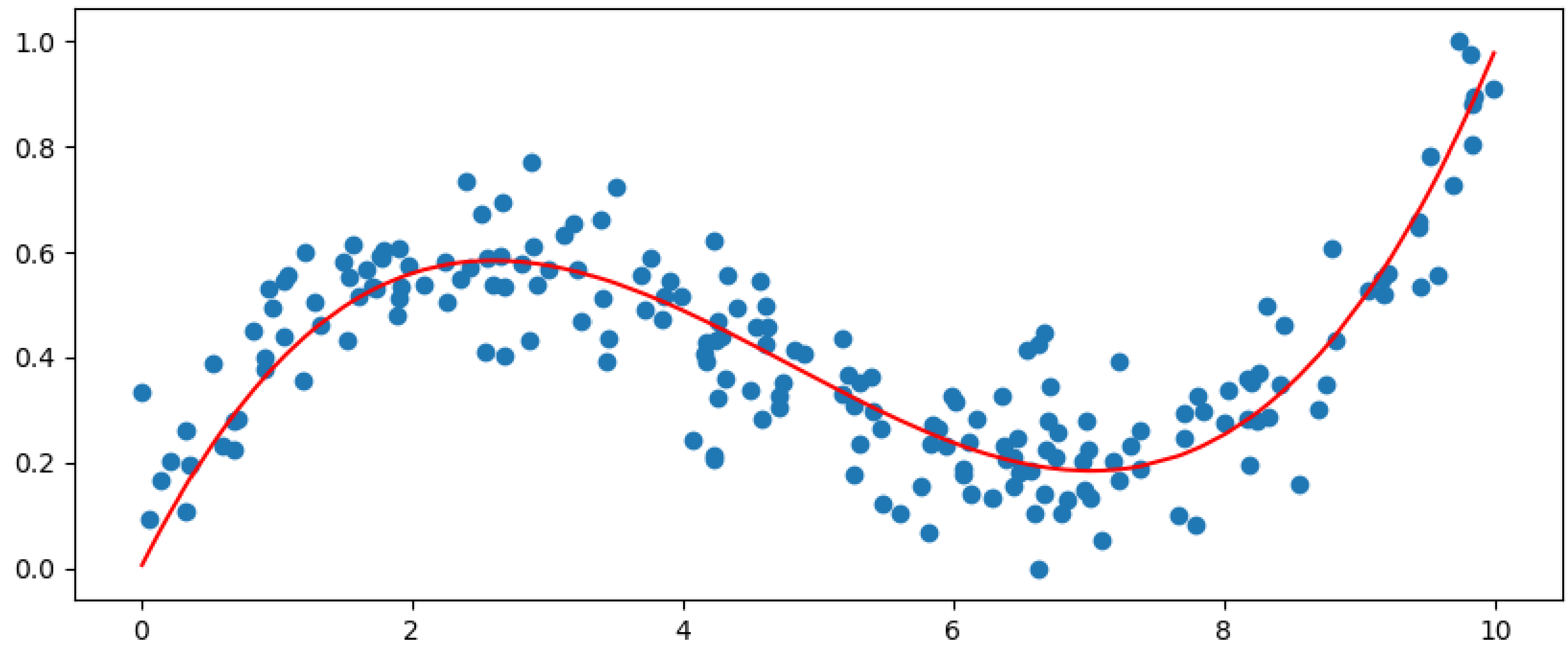
Below we find the cubic polynomial which best fits our data.

In [359]:
```
raw = readcsv("/home/john/Julia/xy_data.csv")
width = 4
x = raw[:,1]
y = raw[:,2]
T = length(x)
A = zeros(T,width)  #width of 4 for cubic poly
for i = 1:T
    for j = 1:width
        A[i,j] = x[i]^(4 - j)
    end
end
```

In [353]:
```
m = Model(solver=GurobiSolver())
@variable(m, sol[1:width])
@constraint(m, sol[4] == 0) #a zero shift to ensure implication constraint
@objective(m, Min, sum(((A*sol - y)[i])^2 for i = 1:T))
solve(m)
yest = A*getvalue(sol);
```

```
Optimize a model with 1 rows, 4 columns and 1 nonzeros
Model has 10 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e+02, 4e+04]
  QObjective range [4e+02, 5e+07]
  Bounds range     [0e+00, 0e+00]
  RHS range        [0e+00, 0e+00]
Presolve removed 1 rows and 1 columns
Presolve time: 0.00s
Presolved: 0 rows, 3 columns, 0 nonzeros
Presolved model has 6 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 5
 AA' NZ     : 1.000e+00
 Factor NZ  : 3.000e+00
 Factor Ops : 5.000e+00 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   0.00000000e+00  0.00000000e+00  0.00e+00 3.87e+04  0.00e+00     0s
   1  -1.29931662e-01 -1.11441746e-04  1.73e-08 3.87e+04  0.00e+00     0s
   2  -4.91220903e-01 -1.60752391e-03  3.80e-08 3.85e+04  0.00e+00     0s
   3  -7.54239065e-01 -3.79341440e-03  8.33e-08 3.83e+04  0.00e+00     0s
   4  -4.35367036e+00 -1.33657720e-01  1.79e-07 3.64e+04  0.00e+00     0s
   5  -1.77007350e+01 -2.79881916e+00  7.58e-08 2.82e+04  0.00e+00     0s
   6  -1.94466972e+01 -3.50819609e+00  1.93e-08 2.69e+04  0.00e+00     0s
   7  -2.03507471e+01 -3.92241723e+00  4.37e-08 2.62e+04  0.00e+00     0s
   8  -2.07536520e+01 -4.11735339e+00  9.62e-08 2.59e+04  0.00e+00     0s
   9  -2.25060396e+01 -5.05462457e+00  2.06e-07 2.45e+04  0.00e+00     0s
  10  -2.40679264e+01 -6.02124147e+00  4.43e-07 2.32e+04  0.00e+00     0s
  11  -2.57178217e+01 -7.19771102e+00  9.47e-07 2.18e+04  0.00e+00     0s
  12  -3.75882047e+01 -3.75472400e+01  1.06e-06 2.18e-02  0.00e+00     0s
  13  -3.75472825e+01 -3.75472824e+01  3.99e-12 2.18e-08  0.00e+00     0s

Barrier solved model in 13 iterations and 0.00 seconds
Optimal objective -3.75472825e+01
```

In [360]:
```
figure(figsize=(10,4))
scatter(x,y)
plot(x,yest,"r-")
cubic_err = getobjectivevalue(m);
```

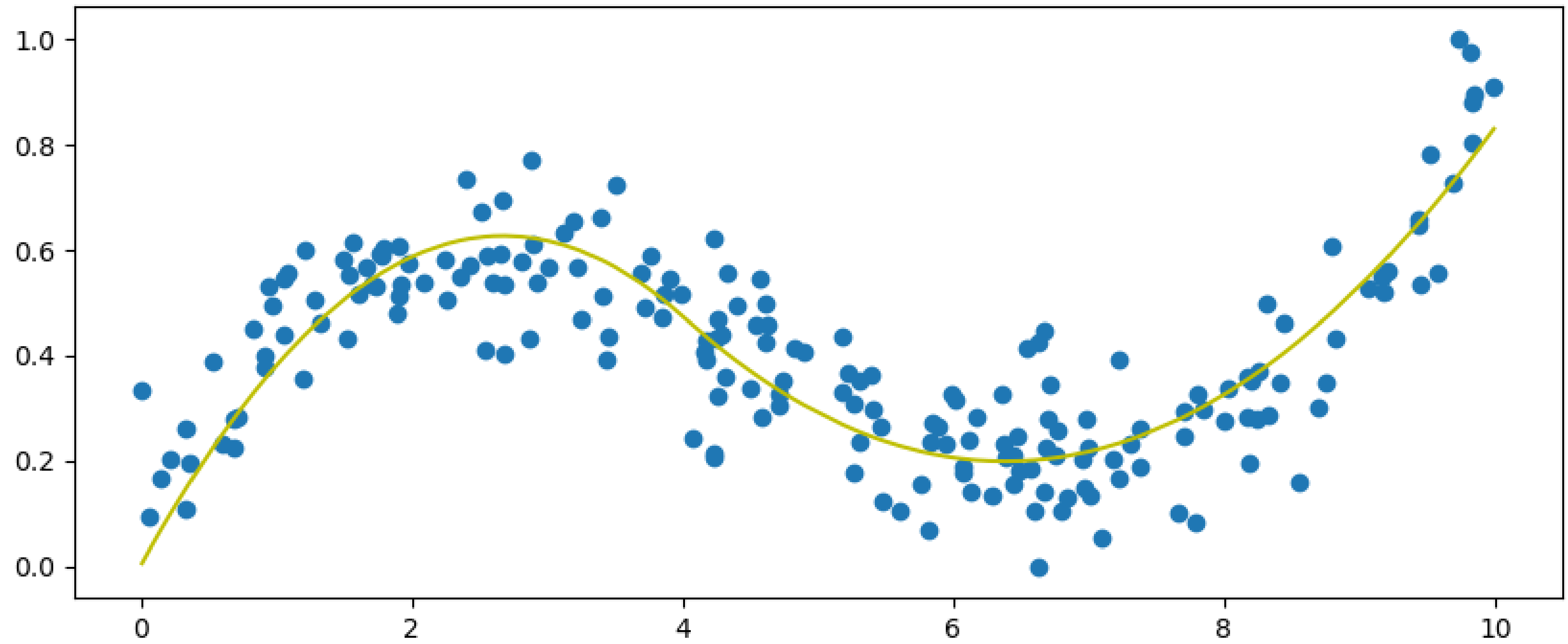


Now we use a piecewise function to model our data.

In [355]:
```
m = Model(solver=GurobiSolver())

width = 3
A = zeros(T,width)

for i = 1:T
    for j = 1:width
        A[i,j] = x[i]^(width - j)
    end
end

@variable(m, pvar[1:3])
@variable(m, qvar[1:3])

@constraint(m, pvar[3] == 0)
@constraint(m, 16pvar[1] + 4pvar[2] + pvar[3] == 16qvar[1] + 4qvar[2] + qvar[3])
@constraint(m, 8pvar[1] + pvar[2] == 8qvar[1] + qvar[2])

@objective(m, Min, sum(((A*pvar - y)[i])^2 for i = 1:76) + sum(((A*qvar - y)[i])^2 for i = 77:T))
solve(m)
spline_err = getobjectivevalue(m);
```

```
Optimize a model with 3 rows, 6 columns and 11 nonzeros
Model has 12 quadratic objective terms
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
  Objective range  [7e+01, 5e+03]
  QObjective range [2e+02, 7e+05]
  Bounds range     [0e+00, 0e+00]
  RHS range        [0e+00, 0e+00]
Presolve removed 1 rows and 1 columns
Presolve time: 0.00s
Presolved: 2 rows, 5 columns, 9 nonzeros
Presolved model has 9 quadratic objective terms
Ordering time: 0.00s

Barrier statistics:
 Free vars  : 8
 AA' NZ     : 8.000e+00
 Factor NZ  : 1.500e+01
 Factor Ops : 5.500e+01 (less than 1 second per iteration)
 Threads    : 1

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   0.00000000e+00  0.00000000e+00  0.00e+00 4.61e+03  0.00e+00     0s
   1  -1.63570631e+00 -1.83031578e-02  1.77e-08 4.51e+03  0.00e+00     0s
   2  -2.64145055e+00 -4.84069233e-02  3.94e-08 4.44e+03  0.00e+00     0s
   3  -7.19991515e+00 -3.84871915e-01  4.65e-08 4.14e+03  0.00e+00     0s
   4  -9.76697661e+00 -7.38357090e-01  7.71e-08 3.96e+03  0.00e+00     0s
   5  -1.34200049e+01 -1.48662734e+00  1.89e-07 3.69e+03  0.00e+00     0s
   6  -2.83338539e+01 -9.65381443e+00  2.79e-07 2.27e+03  0.00e+00     0s
   7  -3.61011676e+01 -2.48640245e+01  3.85e-07 8.49e+02  0.00e+00     0s
   8  -3.73699921e+01 -3.73695143e+01  1.35e-07 8.49e-04  0.00e+00     0s
   9  -3.73695289e+01 -3.73695289e+01  2.14e-13 8.49e-10  0.00e+00     0s

Barrier solved model in 9 iterations and 0.00 seconds
Optimal objective -3.73695289e+01
```

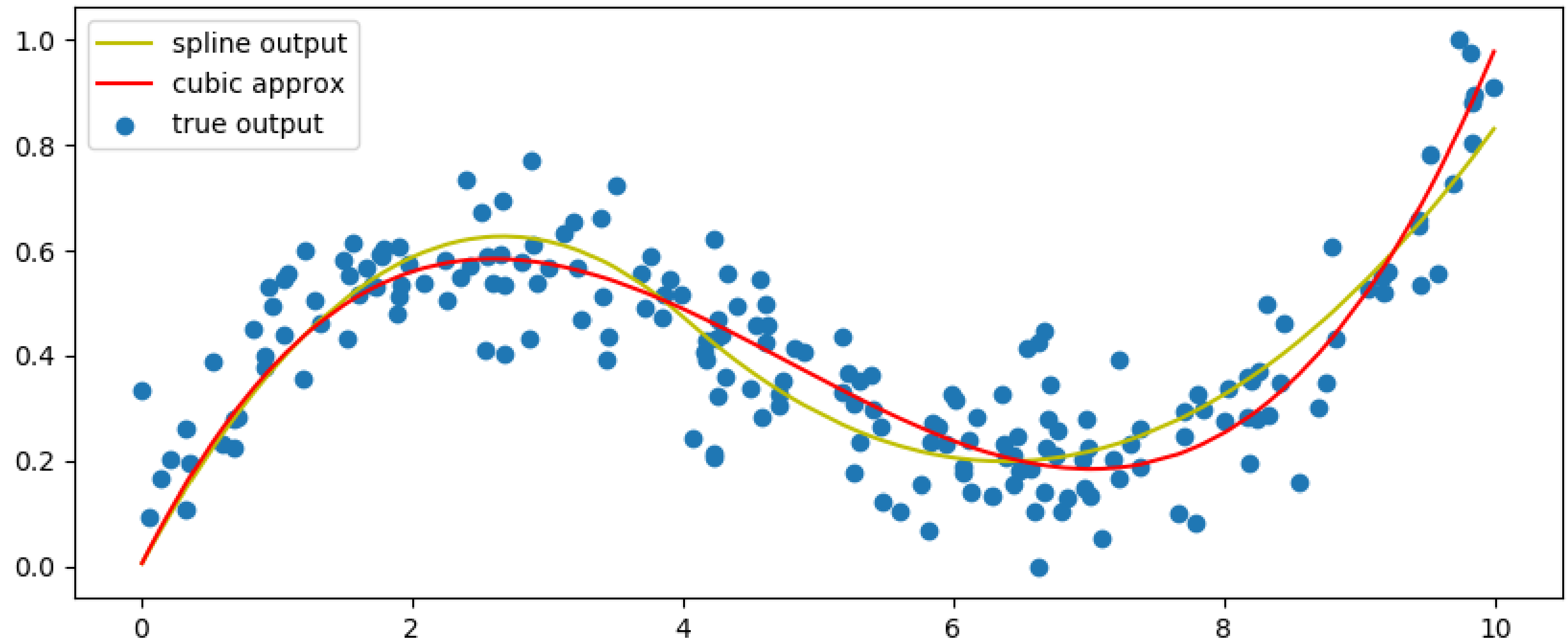
In [356]:
```
out[1:76]

p = getvalue(pvar)
q = getvalue(qvar)

for i = 1:76
   out[i] = p[1]*x[i]^2 + p[2]*x[i]
end

for i = 77:T
    out[i] = q[1]*x[i]^2 + q[2]*x[i] + q[3]
end

figure(figsize=(10,4))
scatter(x,y)
plot(x,out,"y-")
```



Out[356]:
```
1-element Array{Any,1}:
 PyObject <matplotlib.lines.Line2D object at 0x7f4d10cddf10>
```

Comparison of Methods

In [357]:
```
figure(figsize=(10,4))
scatter(x,y)
plot(x,out,"y-")
plot(x,yest,"r-")
legend(["spline output", "cubic approx", "true output"], loc="upper left")

println("Error of cubic model:  ", cubic_err)
println("Error of spline model: ", spline_err)
```




```
Error of cubic model:  1.8806614807652764
Error of spline model: 2.05841510845044
```

I was a bit surprised that the cubic model had a better approximation than the spline model, as I would have thought that because we have essentially two seperate functions for two halves of the graph that there would have been more freedom for the variables in the spline method. Maybe if we had used higher degree polynomials for the spline method (cubics perhaps) it would have produced better results.