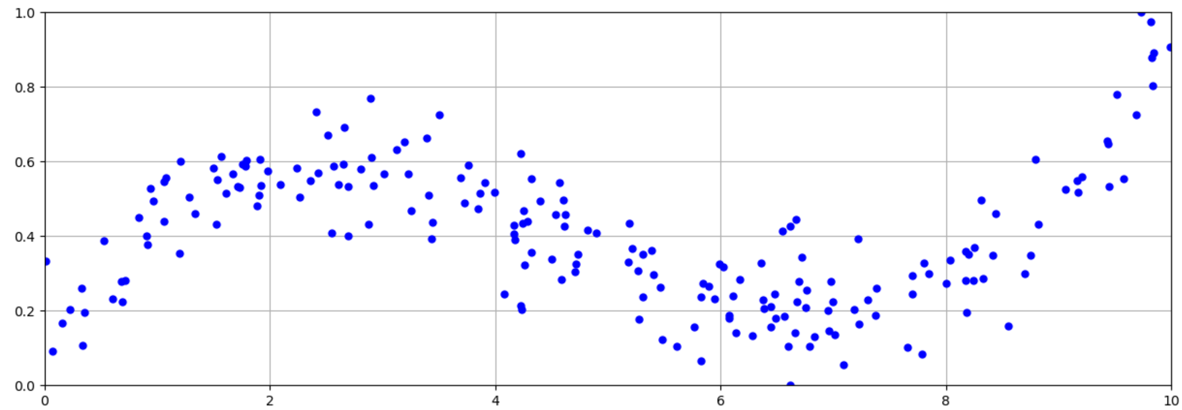
In [44]:

```
#question 1

using CSV
raw = CSV.read("xy_data.csv";header=["x","y"])
x = raw.x;
y = raw.y;
```

In [45]:

```
using PyPlot
figure(figsize=(15,5))
plot(x,y,"b.",markersize = 10)
axis([0,10,0,1])
grid("true")
```




```
C:\Users\yuchi\.juliapro\JuliaPro_v1.0.3.1\conda\3\lib\site-packages\matplotlib\cboo
k\__init__.py:424: MatplotlibDeprecationWarning:
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an actua
l boolean (True/False) instead.
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "
```

In [46]:

```
# order of polynomial to use
k = 3

# 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
```

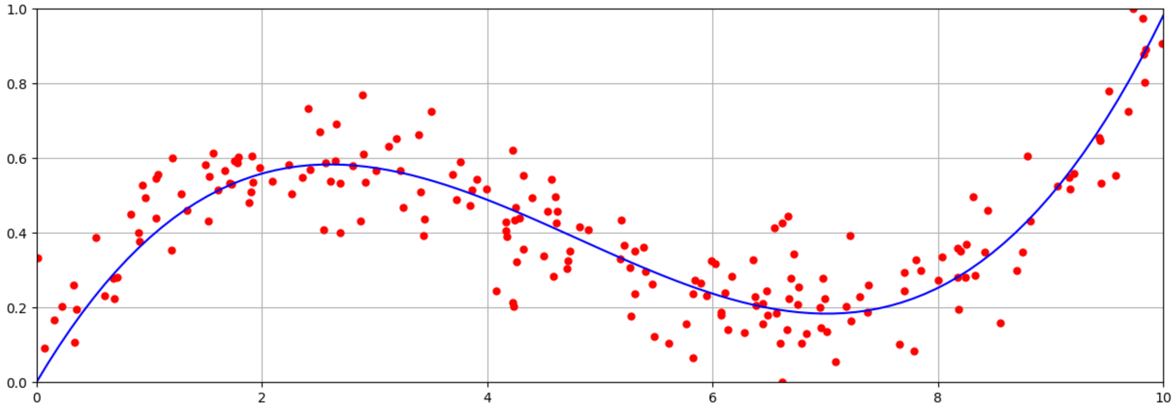
In [47]:

```
# Solve the LEAST SQUARES polynomial fit

using JuMP, Gurobi

#m = Model(solver=MosekSolver(LOG=0))
m = Model(solver=GurobiSolver(OutputFlag=0,BarHomogeneous=1))
#m = Model(solver=GurobiSolver(OutputFlag=0,NumericFocus=3,BarHomogeneous=1))

@variable(m, u[1:k+1])
@constraint(m, u[4] == 0)
@objective(m, Min, 1/1000*sum( (y - A*u).^2 ) )

status = solve(m)
uopt = getvalue(u)
println()
println("Answers for Question 1 a")
println()
println(status)
println(getobjectivevalue(m))
println(uopt)
```

```
Academic license - for non-commercial use only

Answers for Question 1 a

Optimal
0.0018806616352277927
[0.00932501, -0.134546, 0.511155, -0.0]
```

In [48]:

```
#question 1a

using PyPlot
npts = 100
xfine = range(0,stop=10,length=npts)
ffine = ones(npts)
for j = 1:k
    ffine = [ffine.*xfine ones(npts)]
end
yfine = ffine * uopt
figure(figsize=(15,5))
plot( x, y, "r.", markersize=10)
plot( xfine, yfine, "b-")
axis([0,10,0,1])
grid()
```



In [49]:

```
#question 1b

# order of polynomial to use
k = 2

n = length(x)
A2 = zeros(n,2*k+2)
for i = 1:n
    for j = 1:(k+1)*2
        if (0 <= x[i] <4)
            if(1<=j<=3)
                A2[i,j] = x[i]^(k+1-j);
            end
        elseif (4 <= x[i] <=10)
            if(3<j)
                A2[i,j] = x[i]^(k+4-j);
            end
        end
    end
end
```

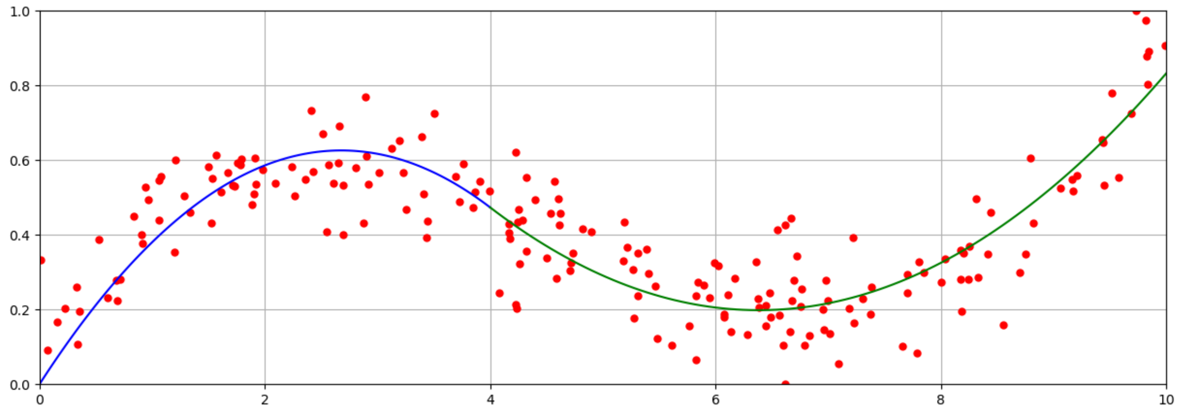
In [50]:

```
using JuMP, Gurobi

#m = Model(solver=MosekSolver(LOG=0))
m1b = Model(solver=GurobiSolver(OutputFlag=0,BarHomogeneous=1))
#m = Model(solver=GurobiSolver(OutputFlag=0,NumericFocus=3,BarHomogeneous=1))

@variable(m1b, u[1:(k+1)*2])
@constraint(m1b, u[3] == 0)
@constraint(m1b,(16*u[1]+4*u[2]+u[3])==(16*u[4]+4*u[5]+u[6]))
@constraint(m1b,(8*u[1]+u[2])==(8*u[4]+u[5]))  ##maintain the slop equal
@objective(m1b, Min, 1/1000*sum( (y - A2*u).^2 ) )

status = solve(m1b)
uopt = getvalue(u)
println()
println("Answers for Question 1 b")
println()
println(status)
println(getobjectivevalue(m1b))
println(uopt)
```

```
Academic license - for non-commercial use only

Answers for Question 1 b

Optimal
0.002058415109440985
[-0.0873261, 0.467682, -0.0, 0.0484683, -0.618673, 2.17271]
```

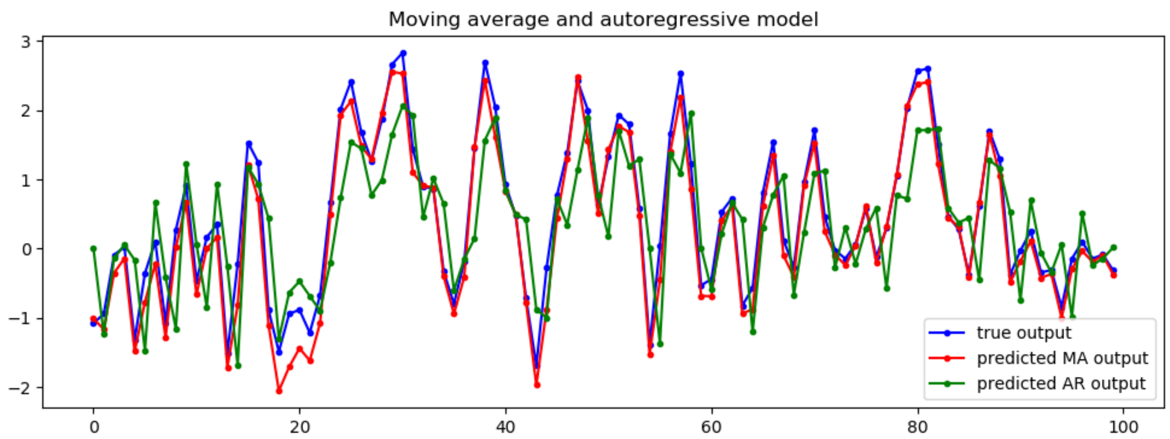
In [51]:

```
#question 1a

using PyPlot
npts = 100
xfine1 = range(0,stop=4,length=npts)
ffine1 = ones(npts)
for j = 1:k
    ffine1 = [ffine1.*xfine1 ones(npts)]
end
yfine1 = ffine1 * uopt[1:3]
xfine2 = range(4,stop=10,length=npts)
ffine2 = ones(npts)
for j = 1:k
    ffine2 = [ffine2.*xfine2 ones(npts)]
end
yfine2 = ffine2 * uopt[4:6]
figure(figsize=(15,5))
plot( x, y, "r.", markersize=10)
plot( xfine1, yfine1, "b-",xfine2, yfine2, "g-")
axis([0,10,0,1])
grid()
```



In [32]:

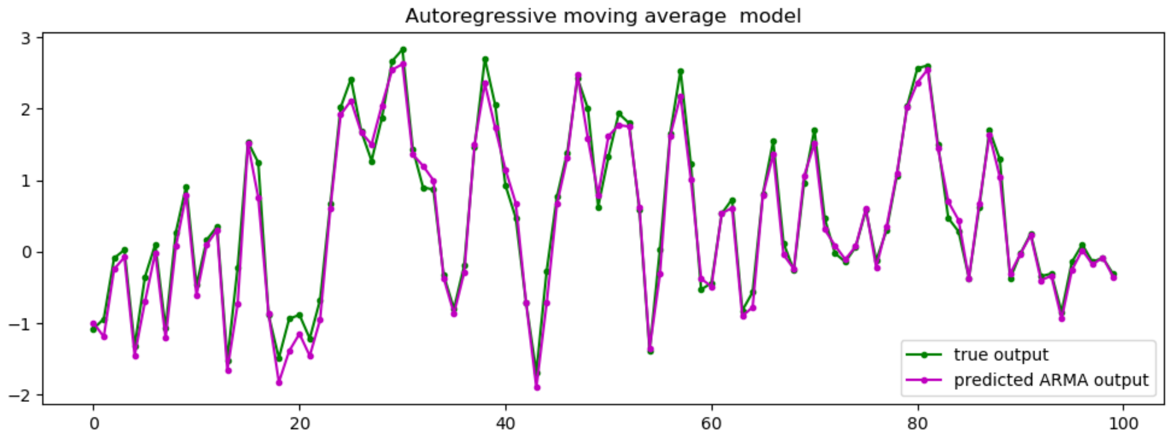
```
#question 2a

# Load the data file (ref: Boyd/263)
using CSV
raw = CSV.read("uy_data.csv";header=["u","y"]);
u = raw[:,1];
y = raw[:,2];
T = length(u)

using  PyPlot, LinearAlgebra

# generate A matrix. Using more width creates better fit.   (MA model)
width = 5
AMA = zeros(T,width)
for i = 1:width
    AMA[i:end,i] = u[1:end-i+1]
end
woptMA = AMA\y
yestMA = AMA*woptMA

# generate A matrix. Using more width creates better fit.   (AR model)
width = 5
AAR = zeros(T,width)
for i = 1:width
    AAR[i+1:end,i] = y[1:end-i]
end
woptAR = AAR\y
yestAR = AAR*woptAR

figure(figsize=(12,4))
plot(y,"b.-",yestMA,"r.-",yestAR,"g.-")
legend(["true output", "predicted MA output","predicted AR output"], loc="lower right");
title("Moving average and autoregressive model");
println()
println("MA:    ",LinearAlgebra.norm(yestMA-y))
println("AR:    ",LinearAlgebra.norm(yestAR-y))
```




```
MA:    2.460854388269911
AR:    7.436691765656794
```

In [35]:

```
#question 2a

using LinearAlgebra

# generate A matrix. Using more width creates better fit.  (MA model)
width = 1
A = zeros(T,2*width)
for i = 1:width
    A[i:end,i] = u[1:end-i+1]
end
for i = 1:width
    A[i+1:end,i+1] = y[1:end-i]
end
wopt = A\y
yest = A*wopt

figure(figsize=(12,4))
plot(y,"g.-",yest,"m.-")
legend(["true output", "predicted ARMA output"], loc="lower right");
title("Autoregressive moving average  model");
println()
println("ARMA:   ",LinearAlgebra.norm(yest-y))
```




```
ARMA:   1.8565828148734604
```

In [36]:

```
#question 2c

println("MA:   ",LinearAlgebra.norm(yestMA-y))
println("AR:   ",LinearAlgebra.norm(yestAR-y))
println("ARMA:   ",LinearAlgebra.norm(yest-y))
```

```
MA:   2.460854388269911
AR:   7.436691765656794
ARMA:   1.8565828148734604
```

Since ARMA is the smallest, ARMA model is better than orther two models because it has the smallest error.
Also, AR is the largerest, so AR is worse than other two models because it has the largest error.

In [38]:

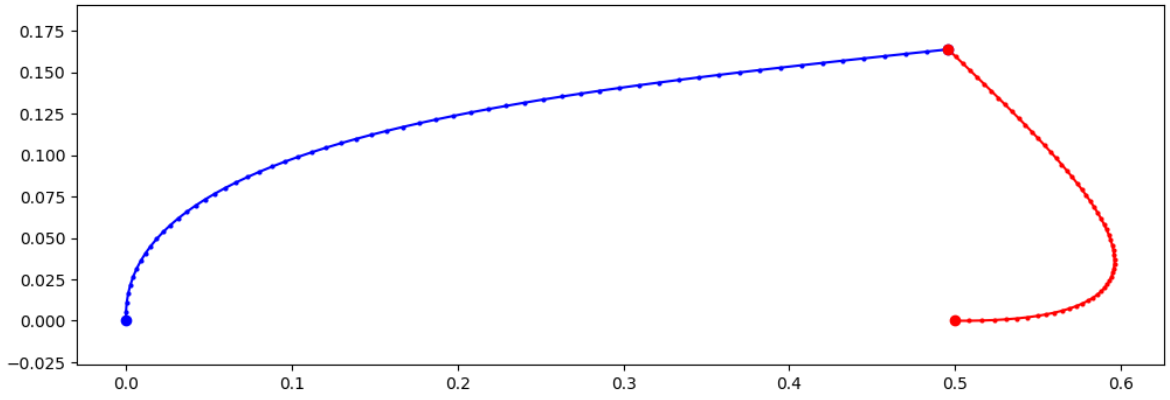
```
#question 3a

using JuMP, Gurobi

    k = 2              # number of waypoints
    T = zeros(Int,k)   # vector of timepoints

    T[1] = 1
    T[2] = 60

    m = Model(solver = GurobiSolver(OutputFlag = 0))

    @variable(m, x1[1:2,1:T[k]])  # resulting position
    @variable(m, v1[1:2,1:T[k]])  # resulting velocity
    @variable(m, u1[1:2,1:T[k]])  # thruster input
    @variable(m, x2[1:2,1:T[k]])  # resulting position
    @variable(m, v2[1:2,1:T[k]])  # resulting velocity
    @variable(m, u2[1:2,1:T[k]])  # thruster input

    # satisfy the dynamics (with zero initial velocity)
    @constraint(m, v1[:,1] .== [0;20])
    @constraint(m, v2[:,1] .== [30;0])

    #satisfy the position constrants
    @constraint(m, x1[:,60] .== x2[:,60])
    @constraint(m, x1[:,1] .== [0,0])
    @constraint(m, x2[:,1] .== [0.5,0])

    for t in 1:T[k]-1
        @constraint(m, x1[:,t+1] .== x1[:,t] + (1/3600)*v1[:,t])
        @constraint(m, v1[:,t+1] .== v1[:,t] + u1[:,t])
        @constraint(m, x2[:,t+1] .== x2[:,t] + (1/3600)*v2[:,t])
        @constraint(m, v2[:,t+1] .== v2[:,t] + u2[:,t])
    end

    @objective(m,Min,sum(u1.^2) + sum(u2.^2))
    solve(m)

println("Minimized total energy: ",getobjectivevalue(m))
println()
println("Sequence of thruster of Alice: ")
println(getvalue(u1))
println("Sequence of thruster of Bog: ")
println(getvalue(u2))
println()
println("Final position of Alice: ")
println(getvalue(x1[:,60]))
println("Final position of Bog: ")
println(getvalue(x2[:,60]))
```

```
Academic license - for non-commercial use only
Minimized total energy: 105.9307047910204

Sequence of thruster of Alice:
[1.5515 1.52475 1.498 1.47125 1.4445 1.41775 1.391 1.36425 1.3375 1.31075 1.284 1.25
725 1.2305 1.20375 1.177 1.15025 1.1235 1.09675 1.07 1.04325 1.0165 0.98975 0.963 0.
93625 0.9095 0.88275 0.856 0.82925 0.8025 0.77575 0.749 0.72225 0.6955 0.66875 0.642
0.61525 0.5885 0.56175 0.535 0.50825 0.4815 0.45475 0.428 0.40125 0.3745 0.34775 0.3
21 0.29425 0.2675 0.24075 0.214 0.18725 0.1605 0.13375 0.107 0.08025 0.0535 0.02675
```

```
-0.0 -0.0; -0.512821 -0.503979 -0.495137 -0.486295 -0.477454 -0.468612 -0.45977 -0.4
50928 -0.442087 -0.433245 -0.424403 -0.415561 -0.40672 -0.397878 -0.389036 -0.380195
-0.371353 -0.362511 -0.353669 -0.344828 -0.335986 -0.327144 -0.318302 -0.309461 -0.3
00619 -0.291777 -0.282935 -0.274094 -0.265252 -0.25641 -0.247569 -0.238727 -0.229885
-0.221043 -0.212202 -0.20336 -0.194518 -0.185676 -0.176835 -0.167993 -0.159151 -0.15
0309 -0.141468 -0.132626 -0.123784 -0.114943 -0.106101 -0.0972591 -0.0884173 -0.0795
756 -0.0707339 -0.0618921 -0.0530504 -0.0442087 -0.0353669 -0.0265252 -0.0176835 -0.
00884173 -0.0 -0.0]
Sequence of thruster of Bog:
[-1.5515 -1.52475 -1.498 -1.47125 -1.4445 -1.41775 -1.391 -1.36425 -1.3375 -1.31075
-1.284 -1.25725 -1.2305 -1.20375 -1.177 -1.15025 -1.1235 -1.09675 -1.07 -1.04325 -1.
0165 -0.98975 -0.963 -0.93625 -0.9095 -0.88275 -0.856 -0.82925 -0.8025 -0.77575 -0.7
49 -0.72225 -0.6955 -0.66875 -0.642 -0.61525 -0.5885 -0.56175 -0.535 -0.50825 -0.481
5 -0.45475 -0.428 -0.40125 -0.3745 -0.34775 -0.321 -0.29425 -0.2675 -0.24075 -0.214
-0.18725 -0.1605 -0.13375 -0.107 -0.08025 -0.0535 -0.02675 -0.0 -0.0; 0.512821 0.503
979 0.495137 0.486295 0.477454 0.468612 0.45977 0.450928 0.442087 0.433245 0.424403
0.415561 0.40672 0.397878 0.389036 0.380195 0.371353 0.362511 0.353669 0.344828 0.33
5986 0.327144 0.318302 0.309461 0.300619 0.291777 0.282935 0.274094 0.265252 0.25641
0.247569 0.238727 0.229885 0.221043 0.212202 0.20336 0.194518 0.185676 0.176835 0.16
7993 0.159151 0.150309 0.141468 0.132626 0.123784 0.114943 0.106101 0.0972591 0.0884
173 0.0795756 0.0707339 0.0618921 0.0530504 0.0442087 0.0353669 0.0265252 0.0176835
0.00884173 -0.0 -0.0]

Final position of Alice:
[0.495833, 0.163889]
Final position of Bog:
[0.495833, 0.163889]
```

In [40]:

```
using PyPlot
figure(figsize=(12,4))
xopt1 = getvalue(x1)
xopt2 = getvalue(x2)
plot( xopt1[1,:], xopt1[2,:], "b.-", markersize=4 )
plot( xopt2[1,:], xopt2[2,:], "r.-", markersize=4 )
plot( xopt1[1,T], xopt1[2,T], "b.", markersize=12 )
plot( xopt2[1,T], xopt2[2,T], "r.", markersize=12 )
axis("equal")
#axis((1.,6.,-1,3.5));
```



In [41]:

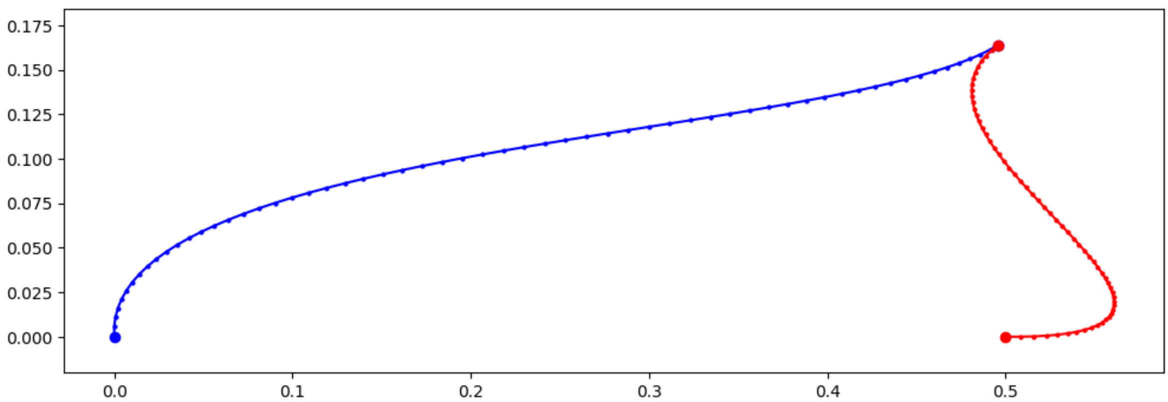
```
#question 3b

using JuMP, Gurobi

    k = 2              # number of waypoints
    T = zeros(Int,k)   # vector of timepoints

    T[1] = 1
    T[2] = 60

    m = Model(solver = GurobiSolver(OutputFlag = 0))

    @variable(m, x1[1:2,1:T[k]])  # resulting position
    @variable(m, v1[1:2,1:T[k]])  # resulting velocity
    @variable(m, u1[1:2,1:T[k]])  # thruster input
    @variable(m, x2[1:2,1:T[k]])  # resulting position
    @variable(m, v2[1:2,1:T[k]])  # resulting velocity
    @variable(m, u2[1:2,1:T[k]])  # thruster input

    # satisfy the dynamics (with zero initial velocity)
    @constraint(m, v1[:,1] .== [0;20])
    @constraint(m, v2[:,1] .== [30;0])

    #satisfy the position constrants
    @constraint(m, x1[:,60] .== x2[:,60])
    @constraint(m, x1[:,1] .== [0,0])
    @constraint(m, x2[:,1] .== [0.5,0])

    #satisfy the final volocity
    @constraint(m,v1[:,60] .== v2[:,60])

    for t in 1:T[k]-1
        @constraint(m, x1[:,t+1] .== x1[:,t] + (1/3600)*v1[:,t])
        @constraint(m, v1[:,t+1] .== v1[:,t] + u1[:,t])
        @constraint(m, x2[:,t+1] .== x2[:,t] + (1/3600)*v2[:,t])
        @constraint(m, v2[:,t+1] .== v2[:,t] + u2[:,t])
    end

    @objective(m,Min,sum(u1.^2) + sum(u2.^2))
    solve(m)

println("Minimized total energy: ",getobjectivevalue(m))
println()
println("Sequence of thruster of Alice: ")
println(getvalue(u1))
println("Sequence of thruster of Bog: ")
println(getvalue(u2))
println()
println("Final position of Alice: ")
println(getvalue(x1[:,60]))
println("Final position of Bog: ")
println(getvalue(x2[:,60]))
```

```
Academic license - for non-commercial use only
Minimized total energy: 234.57042665108122

Sequence of thruster of Alice:
[2.54237 2.46347 2.38457 2.30567 2.22677 2.14787 2.06897 1.99006 1.91116 1.83226 1.7
5336 1.67446 1.59556 1.51666 1.43776 1.35885 1.27995 1.20105 1.12215 1.04325 0.96434
```

```
8 0.885447 0.806546 0.727645 0.648743 0.569842 0.490941 0.41204 0.333139 0.254237 0.
175336 0.0964348 0.0175336 -0.0613676 -0.140269 -0.21917 -0.298071 -0.376973 -0.4558
74 -0.534775 -0.613676 -0.692577 -0.771479 -0.85038 -0.929281 -1.00818 -1.08708 -1.1
6598 -1.24489 -1.32379 -1.40269 -1.48159 -1.56049 -1.63939 -1.71829 -1.79719 -1.8761
-1.955 -2.0339 -0.0; -0.677966 -0.660432 -0.642899 -0.625365 -0.607832 -0.590298 -0.
572764 -0.555231 -0.537697 -0.520164 -0.50263 -0.485096 -0.467563 -0.450029 -0.43249
6 -0.414962 -0.397428 -0.379895 -0.362361 -0.344828 -0.327294 -0.30976 -0.292227 -0.
274693 -0.25716 -0.239626 -0.222092 -0.204559 -0.187025 -0.169492 -0.151958 -0.13442
4 -0.116891 -0.0993571 -0.0818235 -0.0642899 -0.0467563 -0.0292227 -0.0116891 0.0058
4454 0.0233781 0.0409117 0.0584454 0.075979 0.0935126 0.111046 0.12858 0.146113 0.16
3647 0.181181 0.198714 0.216248 0.233781 0.251315 0.268849 0.286382 0.303916 0.32144
9 0.338983 -0.0]
Sequence of thruster of Bog:
[-2.54237 -2.46347 -2.38457 -2.30567 -2.22677 -2.14787 -2.06897 -1.99006 -1.91116 -
1.83226 -1.75336 -1.67446 -1.59556 -1.51666 -1.43776 -1.35885 -1.27995 -1.20105 -1.1
2215 -1.04325 -0.964348 -0.885447 -0.806546 -0.727645 -0.648743 -0.569842 -0.490941
-0.41204 -0.333139 -0.254237 -0.175336 -0.0964348 -0.0175336 0.0613676 0.140269 0.21
917 0.298071 0.376973 0.455874 0.534775 0.613676 0.692577 0.771479 0.85038 0.929281
1.00818 1.08708 1.16598 1.24489 1.32379 1.40269 1.48159 1.56049 1.63939 1.71829 1.79
719 1.8761 1.955 2.0339 -0.0; 0.677966 0.660432 0.642899 0.625365 0.607832 0.590298
0.572764 0.555231 0.537697 0.520164 0.50263 0.485096 0.467563 0.450029 0.432496 0.41
4962 0.397428 0.379895 0.362361 0.344828 0.327294 0.30976 0.292227 0.274693 0.25716
0.239626 0.222092 0.204559 0.187025 0.169492 0.151958 0.134424 0.116891 0.0993571 0.
0818235 0.0642899 0.0467563 0.0292227 0.0116891 -0.00584454 -0.0233781 -0.0409117 -
0.0584454 -0.075979 -0.0935126 -0.111046 -0.12858 -0.146113 -0.163647 -0.181181 -0.1
98714 -0.216248 -0.233781 -0.251315 -0.268849 -0.286382 -0.303916 -0.321449 -0.33898
3 -0.0]

Final position of Alice:
[0.495833, 0.163889]
Final position of Bog:
[0.495833, 0.163889]
```

In [43]:

```julia
using PyPlot
figure(figsize=(12,4))
xopt1 = getvalue(x1)
xopt2 = getvalue(x2)
plot( xopt1[1,:], xopt1[2,:], "b.-", markersize=4 )
plot( xopt2[1,:], xopt2[2,:], "r.-", markersize=4 )
plot( xopt1[1,T], xopt1[2,T], "b.", markersize=12 )
plot( xopt2[1,T], xopt2[2,T], "r.", markersize=12 )
axis("equal");
```



Clearly, the optimal rendezvous location is not different from the one found in problem a.