HW5

March 3, 2018

1 Homework 5: Least squares

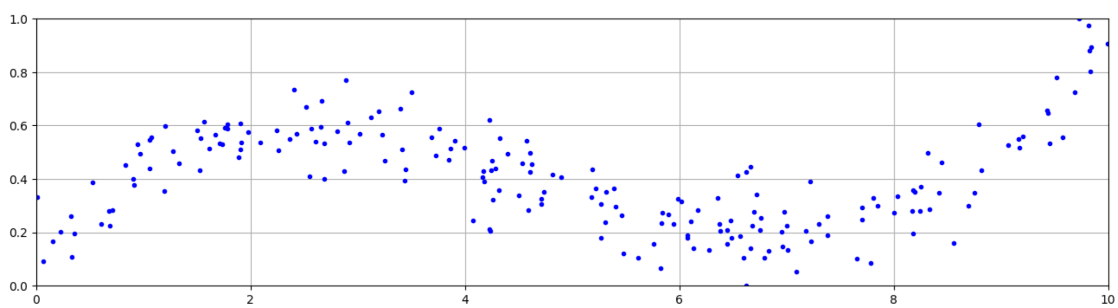
Name: Zihao Qiu
Email: zqiu34@wisc.edu
Campus ID: 9079810942

1.1 1. Spline fitting

1.1.1 (a) Polynomial fit

The data:

```
In [32]: using PyPlot

         input = readcsv("/home/qiuzihao/Desktop/CS524/HW5/xy_data.csv")
         x = input[:, 1]
         y = input[:, 2]

         figure(figsize=(16,4))
         plot(x,y,"b.")
         axis([0,10,0,1])
         grid("on")
```



Use polynomial fit:

```
In [7]: # order of polynomial to use
        k = 3
```

```julia
# fit using a function of the form y=a_1*x^3 + a_2*x^2 + a_3*x  (a_4 = 0)
A = zeros(length(x), 3)
for i = 1:length(x)
    for j = 1:k
        A[i,j] = x[i]^(k+1-j)
    end
end
```

In [9]:
```julia
using JuMP, Gurobi

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

@variable(m, u[1:k])
@objective(m, Min, sum((y-A*u).^2))

status = solve(m)
uopt = getvalue(u)
println(status)
```

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

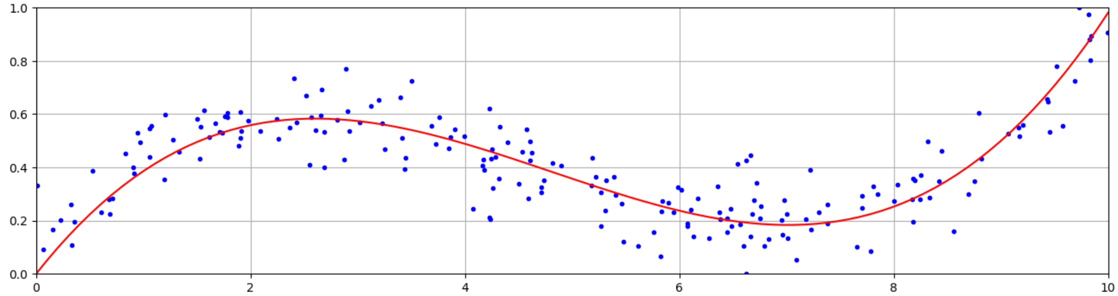
In [10]:
```julia
uopt
```

Out[10]:
```
3-element Array{Float64,1}:
  0.00932501
 -0.134546
  0.511155
```

So the function is as follows:
$y = 0.00932501x^3 - 0.134546x^2 + 0.511155x$
The plot is:

In [31]:
```julia
using PyPlot

npts = 100
xfine = linspace(0,10,npts)
ffine = ones(npts)

for j = 1:k
    ffine = [ffine.*xfine xfine]
end

ffine = ffine[:, 2:k+1]

yfine = ffine * uopt
```

```
figure(figsize=(16,4))
plot(x,y,"b.")
plot(xfine,yfine,"r-")
axis([0,10,0,1])
grid("on")
```



1.1.2 (b) Spline fit

In [47]:
```
# order of polynomial to use
k = 2

num_Ap = 0

for i in 1:length(x)
    if 0<=x[i]<4
        num_Ap = num_Ap + 1
    end
end

# fit using a function of the form y=(p/q)_1*x^2 + (p/q)_2*x + (p/q)_3
Ap = zeros(num_Ap, 3)
for i = 1:num_Ap
    for j = 1:k+1
        if 0<=x[i]<4
            Ap[i,j] = x[i]^(k+1-j)
        end
    end
end

Aq = zeros(length(x)-num_Ap, 3)
for i = 1:length(x)-num_Ap
    for j = 1:k+1
        if 4<=x[i+num_Ap]<10
            Aq[i,j] = x[i+num_Ap]^(k+1-j)
        end
    end
```

```julia
    end

    yp = y[1:num_Ap]
    yq = y[num_Ap+1:length(x)];
```

In [64]:
```julia
using JuMP, Gurobi

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

@variable(m, p[1:k+1])
@variable(m, q[1:k+1])

@constraint(m, p[3] == 0)
@constraint(m, 16p[1]+4p[2]+p[3] == 16q[1]+4q[2]+q[3]) # value equality
@constraint(m, 8p[1]+p[2] == 8q[1]+q[2])              # slope equality

@objective(m, Min, sum((yp-Ap*p).^2) + sum((yq-Aq*q).^2))

status = solve(m)
popt = getvalue(p)
qopt = getvalue(q)
println(status)
```

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

In [65]:
```julia
popt
```

Out[65]:
```
3-element Array{Float64,1}:
 -0.0873261
  0.467682
 -0.0
```

In [66]:
```julia
qopt
```

Out[66]:
```
3-element Array{Float64,1}:
  0.0484683
 -0.618673
  2.17271
```

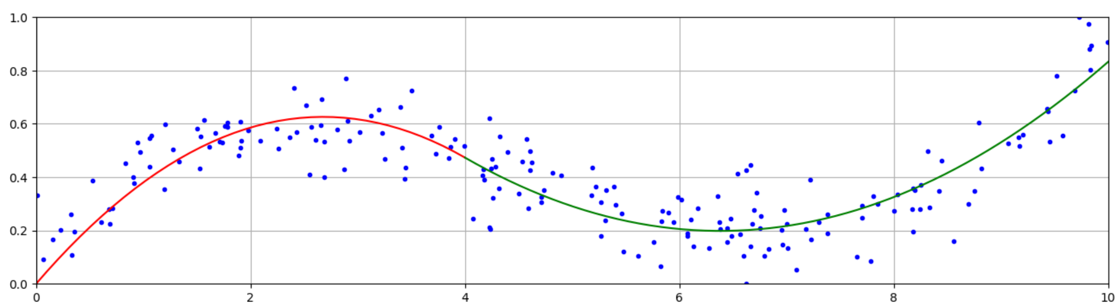
So the function is:

$

$$y = \begin{cases} -0.0873261x^2 + 0.467682x & if \quad 0 \leq x \leq 4 \\ 0.0484683x^2 - 0.618673x + 2.17271 & if \quad 4 \leq x \leq 10 \end{cases} \tag{1}$$

$

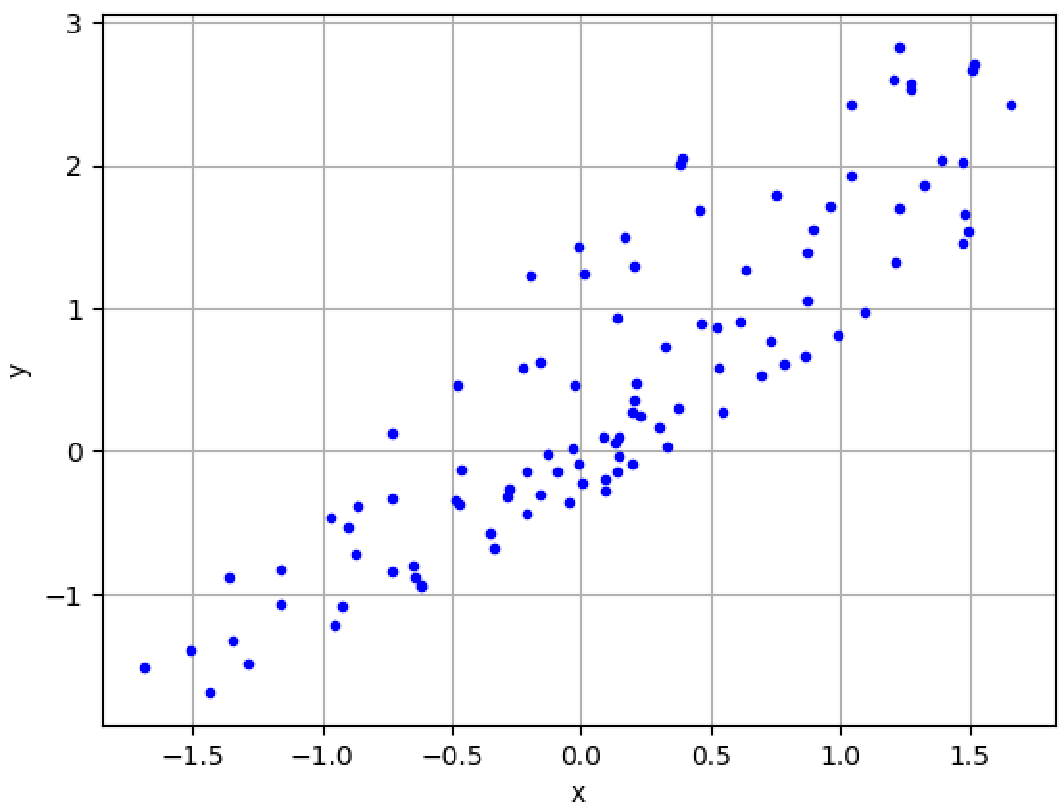
In [67]:
```julia
using PyPlot

### part p
npts_p = 40
xfine_p = linspace(0,4,npts_p)
ffine_p = ones(npts_p)

for j = 1:k
    ffine_p = [ffine_p.*xfine_p ones(npts_p)]
end

yfine_p = ffine_p * popt

### part q
npts_q = 60
xfine_q = linspace(4,10,npts_q)
ffine_q = ones(npts_q)

for j = 1:k
    ffine_q = [ffine_q.*xfine_q ones(npts_q)]
end

yfine_q = ffine_q * qopt

### finally
figure(figsize=(16,4))
plot(x,y,"b.")
plot(xfine_p,yfine_p,"r-")
plot(xfine_q,yfine_q,"g-")
axis([0,10,0,1])
grid("on")
```



1.2 2. Moving averages

1.2.1 (a)

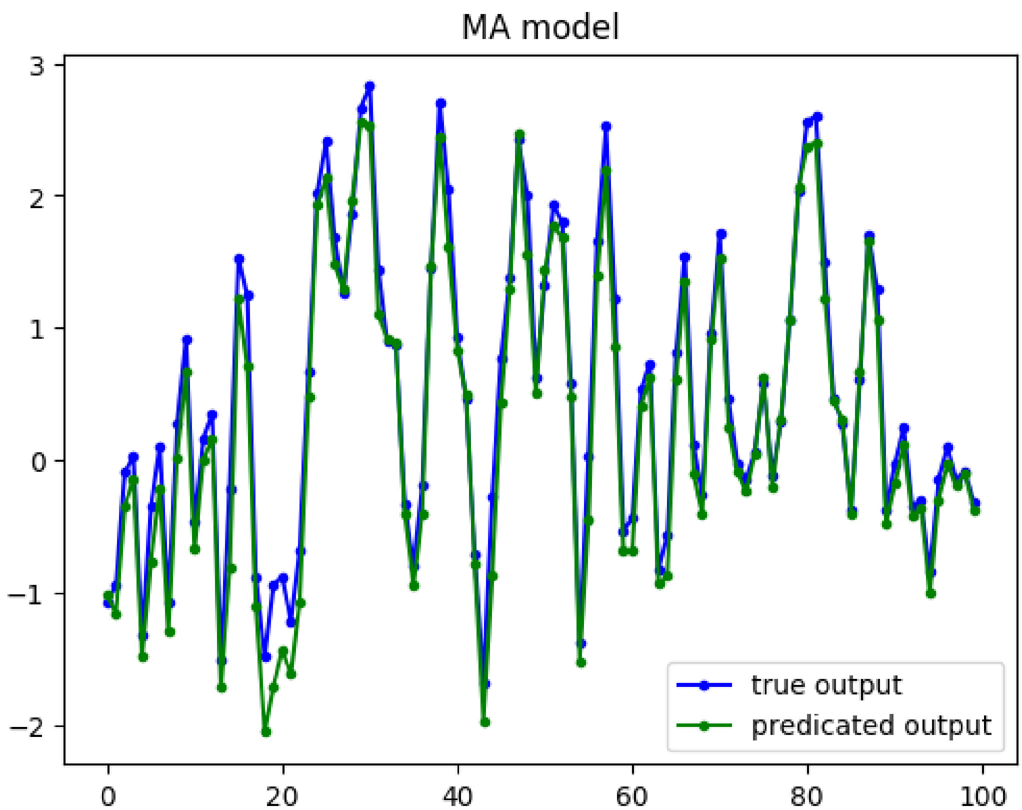
In [24]:
```
using PyPlot

input = readcsv("/home/qiuzihao/Desktop/CS524/HW5/uy_data.csv")
x = input[:, 1]
y = input[:, 2]

plot(x,y,"b.")
xlabel("x")
ylabel("y")
grid("on")
```



First I use MA model:

In [27]:
```
### (MA model)
width = 5
A_MA = zeros(length(x), width)
for i = 1:width
    A_MA[i:end,i] = x[1:end-i+1]
end
```

```
wopt_MA = A_MA\y
yest_MA = A_MA*wopt_MA

plot(y,"b.-",yest_MA,"g.-")
legend(["true output", "predicated output"], loc="lower right")
title("MA model")
println()
println(norm(yest_MA-y))
```

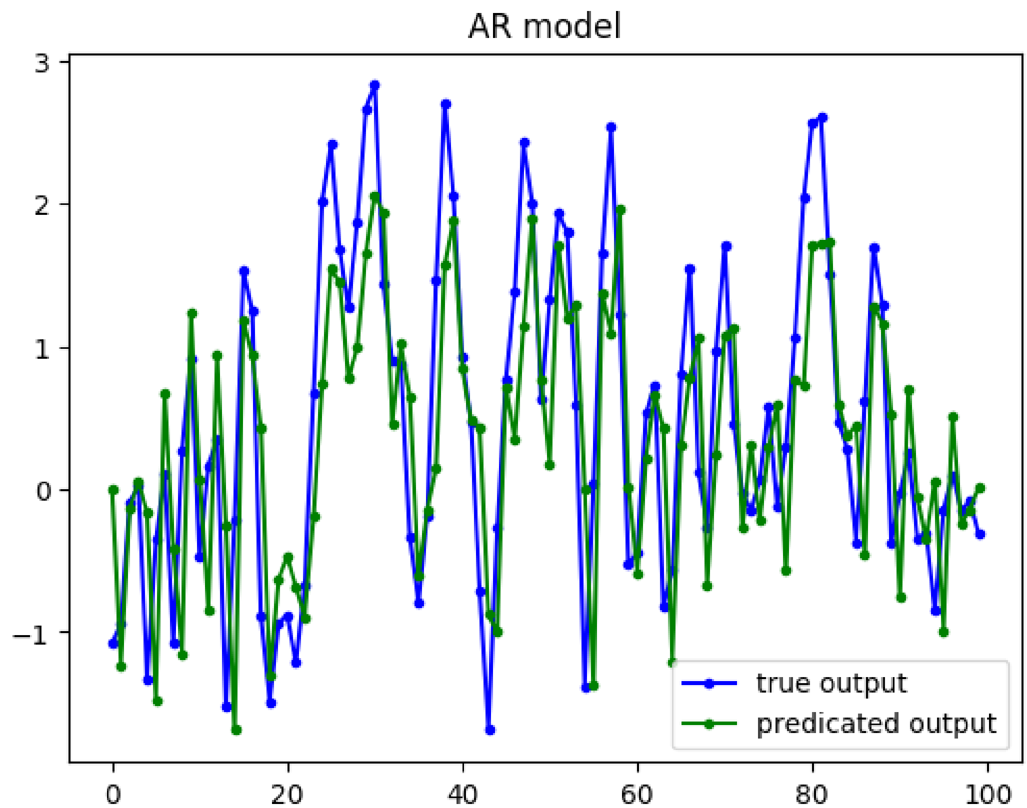



```
2.460854388269911
```

So $\|y - \hat{y}\| = 2.460854388269911$ when using MA model.
Then I use AR model:

```
In [16]: ### (AR model)
         width = 5
         A_AR = zeros(length(y), width)
         for i = 1:width
             A_AR[i+1:end,i] = y[1:end-i]
```

```
end

wopt_AR = A_AR\y
yest_AR = A_AR*wopt_AR

plot(y,"b.-",yest_AR,"g.-")
legend(["true output", "predicated output"], loc="lower right")
title("AR model")
println()
println(norm(yest_AR-y))
```




```
7.436691765656793
```

So $\|y - \hat{y}\| = 7.436691765656793$ when using AR model.

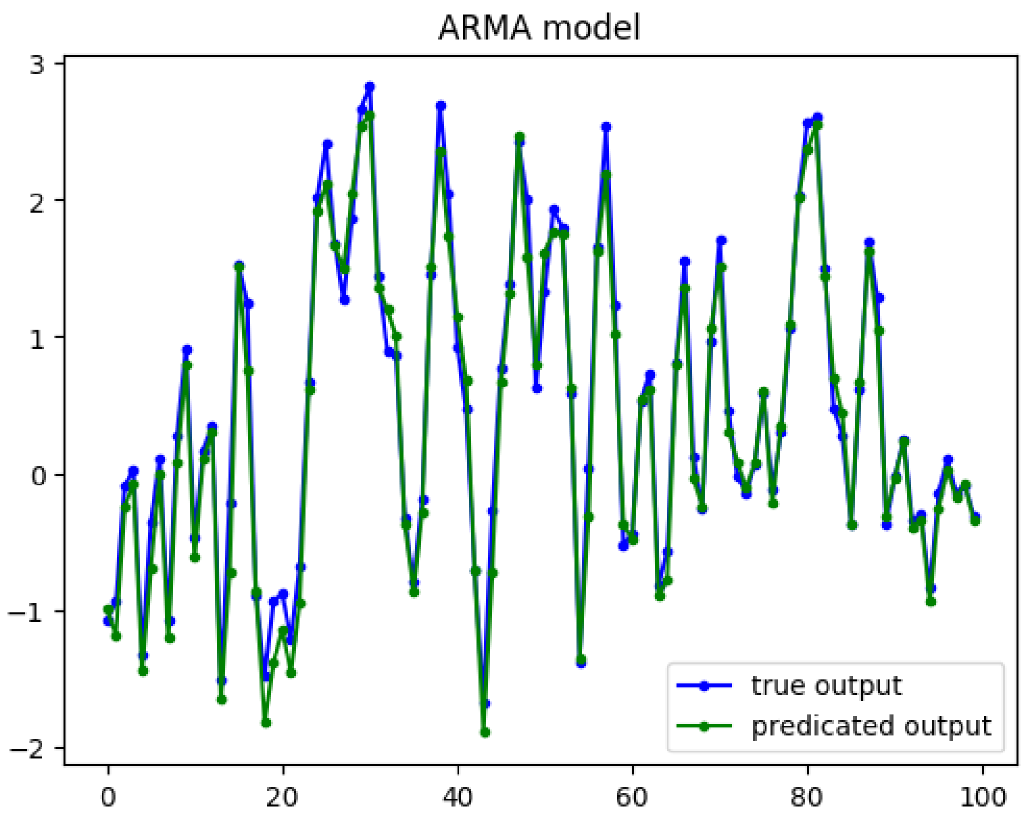
1.2.2 (b)

Because k=l=1, so the ARMA model is: $

$$y_t = a_1 y_{t-1} + b_1 x_t \tag{2}$$

$

In [17]:
```
### (ARMA model)
k = 1
l = 1
A_ARMA = zeros(length(y), k+l)
for i = 1:k
    A_ARMA[i+1:end,i] = y[1:end-i]
end
for i = 1:l
    A_ARMA[i:end,k+i] = x[1:end-i+1]
end

wopt_ARMA = A_ARMA\y
yest_ARMA = A_ARMA*wopt_ARMA

plot(y,"b.-",yest_ARMA,"g.-")
legend(["true output", "predicated output"], loc="lower right")
title("ARMA model")
println()
println(norm(yest_ARMA-y))
```



```
1.8565828148734604
```

So $\|y - \hat{y}\| = 1.8565828148734604$ when using ARMA model.

1.3 3. Hovercraft rendezvous

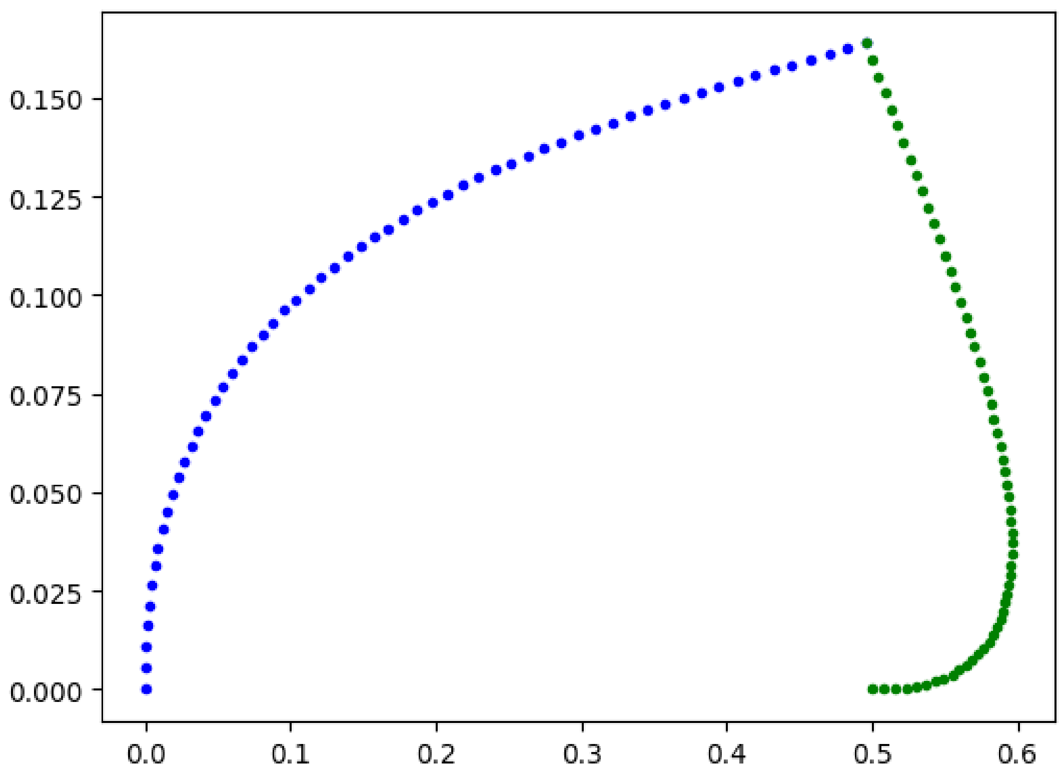
1.3.1 (a)

In [30]:
```
using JuMP, Gurobi

t = 60 # rendezvous at t=60 seconds

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

# for Alice
@variable(m, xA[1:2, 1:t])  # position
@variable(m, vA[1:2, 1:t])  # velocity
@variable(m, uA[1:2, 1:t])  # thrust

# for Bob
@variable(m, xB[1:2, 1:t])  # position
@variable(m, vB[1:2, 1:t])  # velocity
@variable(m, uB[1:2, 1:t])  # thrust

# initial velocity (at t=1)
@constraint(m, vA[:,1] .== [0;20]) # Alice: 20mph north
@constraint(m, vB[:,1] .== [30;0]) # Bob:   30mph east

# initial position (at t=1)
@constraint(m, xA[:,1] .== [0;0])   # Alice: (0, 0)
@constraint(m, xB[:,1] .== [0.5;0]) # Bob:   (0.5, 0)

# rendezvous at t=60
@constraint(m, xA[:,t] .== xB[:,t])

# satisfy the dynamics
for j in 1:t-1
    @constraint(m, xA[:,j+1] .== xA[:,j] + vA[:,j]/3600)
    @constraint(m, vA[:,j+1] .== vA[:,j] + uA[:,j])
    @constraint(m, xB[:,j+1] .== xB[:,j] + vB[:,j]/3600)
    @constraint(m, vB[:,j+1] .== vB[:,j] + uB[:,j])
end

@objective(m, Min, sum(uA.^2) + sum(uB.^2))

solve(m)
```

```
        XA = getvalue(xA)
        XB = getvalue(xB)

        println("||uA||^2: ", getvalue(sum(uA.^2)))
        println("||uB||^2: ", getvalue(sum(uB.^2)))
        println("Alice rendezvous location:", getvalue(xA[:,t]))
        println("Bob rendezvous location:", getvalue(xB[:,t]))
        println("Alice rendezvous velocity:", getvalue(vA[:,t]))
        println("Bob rendezvous velocity:", getvalue(vB[:,t]))
```

```
Academic license - for non-commercial use only
||uA||^2: 52.965352395510195
||uB||^2: 52.965352395510195
Alice rendezvous location:[0.4958333333333333,0.16388888888888897]
Bob rendezvous location:[0.4958333333333333,0.16388888888888897]
Alice rendezvous velocity:[45.769230769230774,4.871794871794866]
Bob rendezvous velocity:[-15.769230769230774,15.128205128205135]
```

```
In [31]: using PyPlot
         plot(XA[1,:], XA[2,:], "b.-")
         plot(XB[1,:], XB[2,:], "g.-");
```



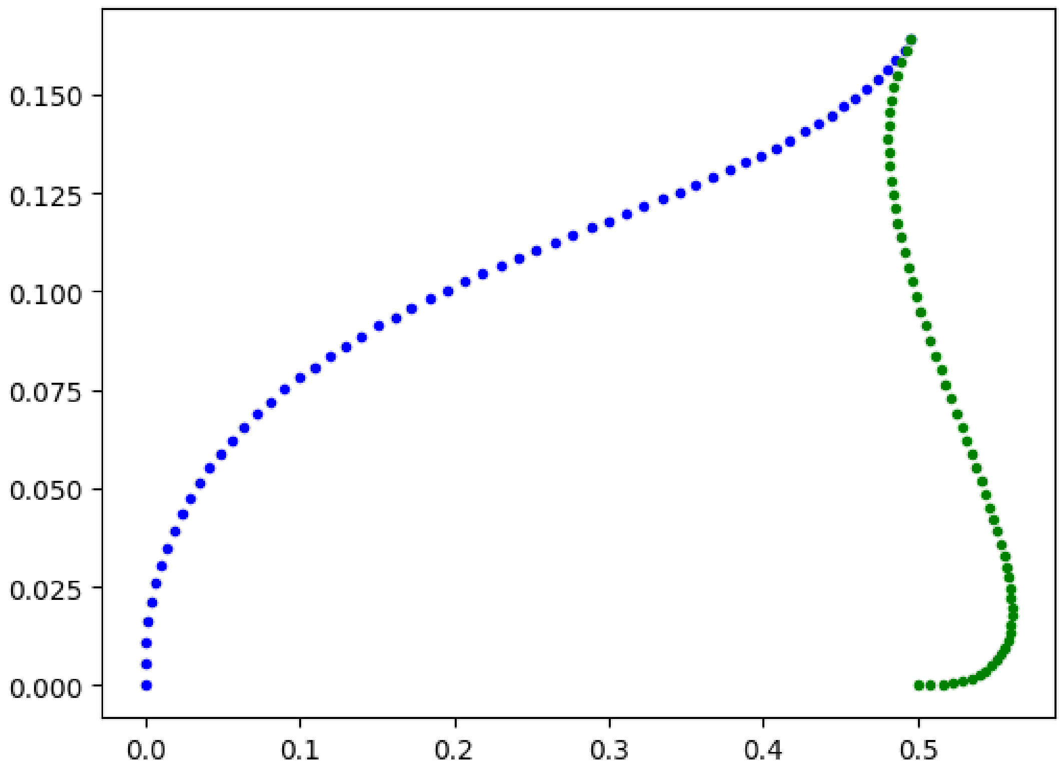
1.3.2 (b)

In [32]:
```
using JuMP, Gurobi

t = 60 # rendezvous at t=60 seconds

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

# for Alice
@variable(m, xA[1:2, 1:t])  # position
@variable(m, vA[1:2, 1:t])  # velocity
@variable(m, uA[1:2, 1:t])  # thrust

# for Bob
@variable(m, xB[1:2, 1:t])  # position
@variable(m, vB[1:2, 1:t])  # velocity
@variable(m, uB[1:2, 1:t])  # thrust

# initial velocity (at t=1)
@constraint(m, vA[:,1] .== [0;20]) # Alice: 20mph north
@constraint(m, vB[:,1] .== [30;0]) # Bob:   30mph east

# initial position (at t=1)
@constraint(m, xA[:,1] .== [0;0])   # Alice: (0, 0)
@constraint(m, xB[:,1] .== [0.5;0]) # Bob:   (0.5, 0)

# rendezvou at t=60
@constraint(m, xA[:,t] .== xB[:,t])
@constraint(m, vA[:,t] .== vB[:, t])

# satisfy the dynamics
for j in 1:t-1
    @constraint(m, xA[:,j+1] .== xA[:,j] + vA[:,j]/3600)
    @constraint(m, vA[:,j+1] .== vA[:,j] + uA[:,j])
    @constraint(m, xB[:,j+1] .== xB[:,j] + vB[:,j]/3600)
    @constraint(m, vB[:,j+1] .== vB[:,j] + uB[:,j])
end

@objective(m, Min, sum(uA.^2) + sum(uB.^2))

solve(m)

XA = getvalue(xA)
XB = getvalue(xB)

println("||uA||^2: ", getvalue(sum(uA.^2)))
println("||uB||^2: ", getvalue(sum(uB.^2)))
println("Alice rendezvous location:", getvalue(xA[:,t]))
```

```
println("Bob rendezvous location:", getvalue(xB[:,t]))
println("Alice rendezvous velocity:", getvalue(vA[:,t]))
println("Bob rendezvous velocity:", getvalue(vB[:,t]))
```

```
Academic license - for non-commercial use only
||uA||^2: 117.28521332554057
||uB||^2: 117.28521332554057
Alice rendezvous location:[0.49583333333333357,0.16388888888888883]
Bob rendezvous location:[0.49583333333333357,0.16388888888888883]
Alice rendezvous velocity:[15.000000000000012,9.999999999999998]
Bob rendezvous velocity:[15.000000000000012,9.999999999999998]
```

In [33]:
```
using PyPlot
plot(XA[1,:], XA[2,:], "b.-")
plot(XB[1,:], XB[2,:], "g.-");
```



We can see that the optimal rendzvous location is the same as the one found in part(a).