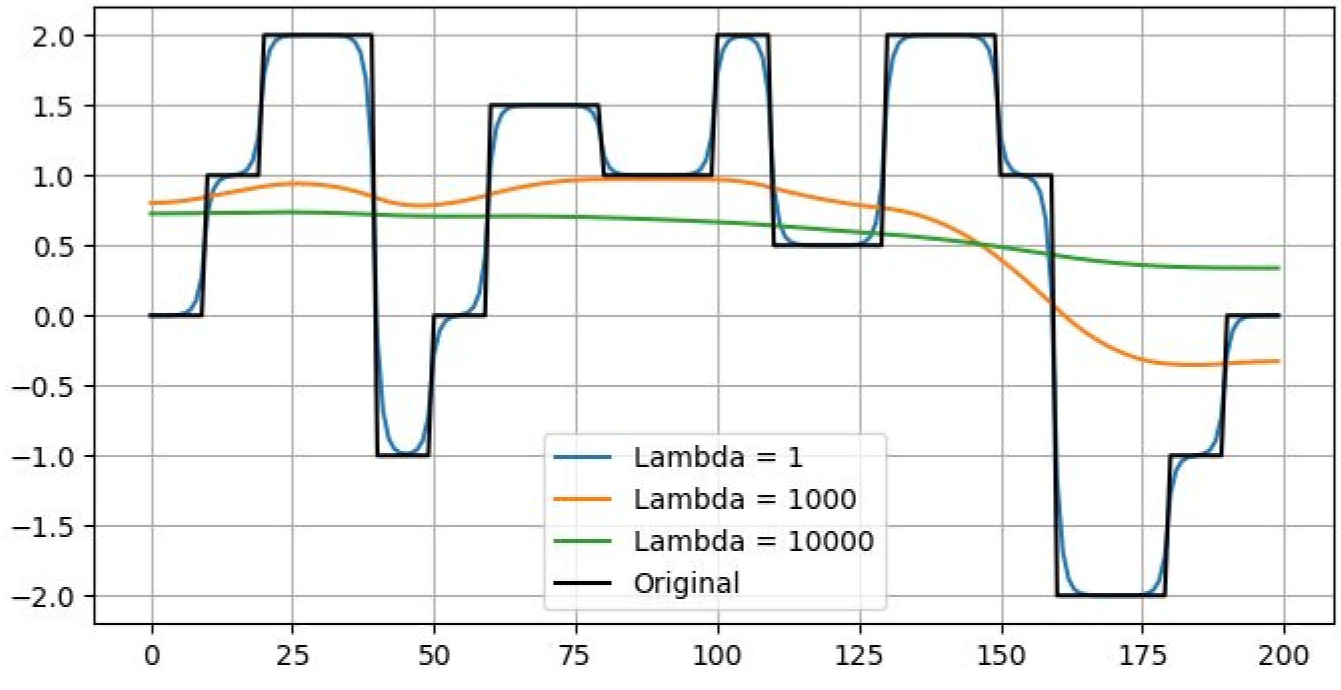
In [1]:

```
#Problem 1
raw = readcsv("voltages.csv");
v = raw[:,1]
T = length(v)

using JuMP, Gurobi, Mosek

lambda=[1, 1000, 10000]
xopt=[]

m = Model(solver = MosekSolver(LOG=0))
@variable(m, x[1:T])

for i in lambda
    @objective(m, Min, sum((x - v).^2) + i*sum((x[j+1] -x[j])^2 for j in 1:199))
    solve(m)
    push!(xopt,getvalue(x))
    end

using PyPlot
figure(figsize=(8,4))

for i in 1:length(xopt)
    plot(xopt[i], label = "Lambda = $(lambda[i])")
end

plot(v, "k", label = "Original")
legend()
grid("on")
```

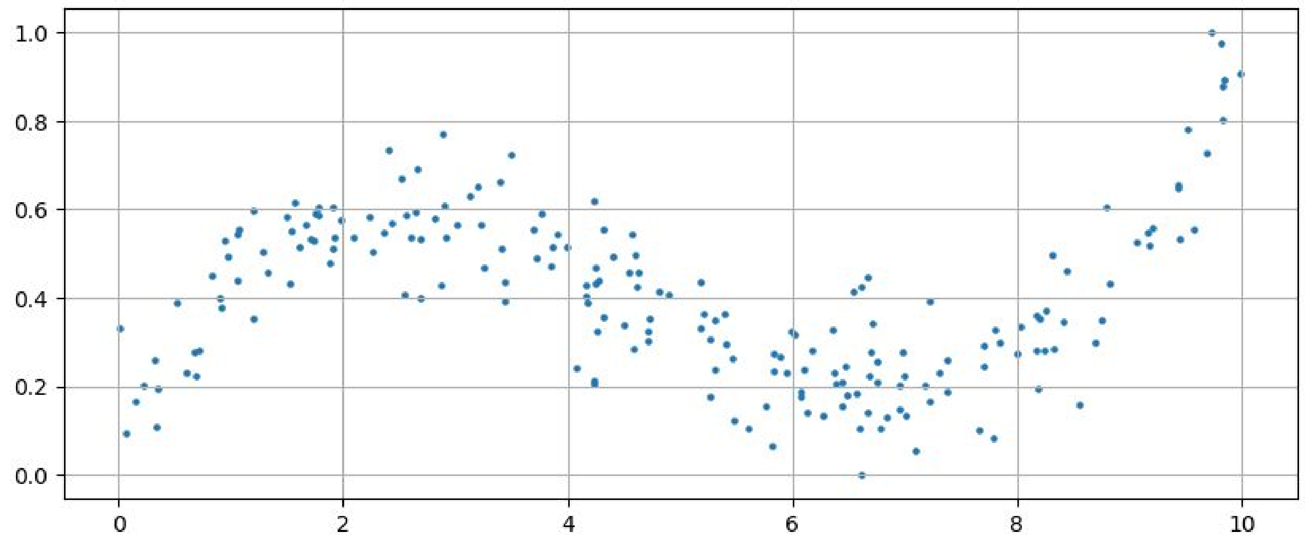


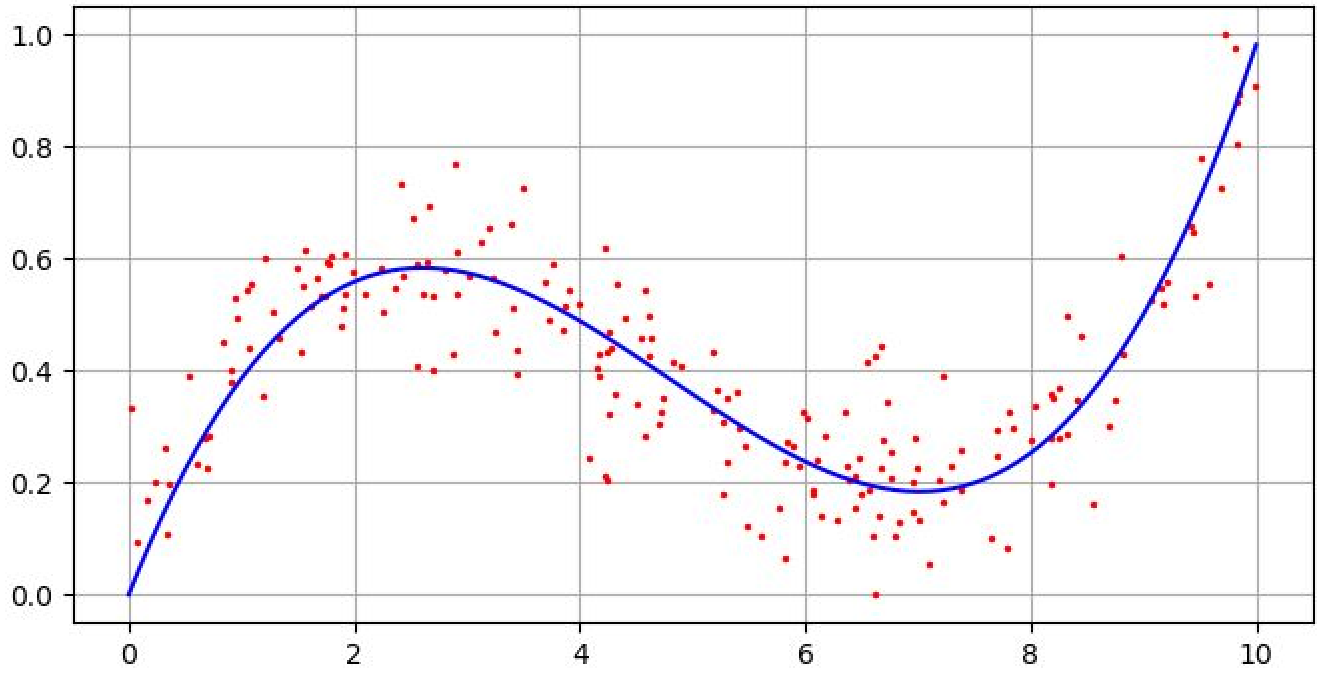
In [2]:

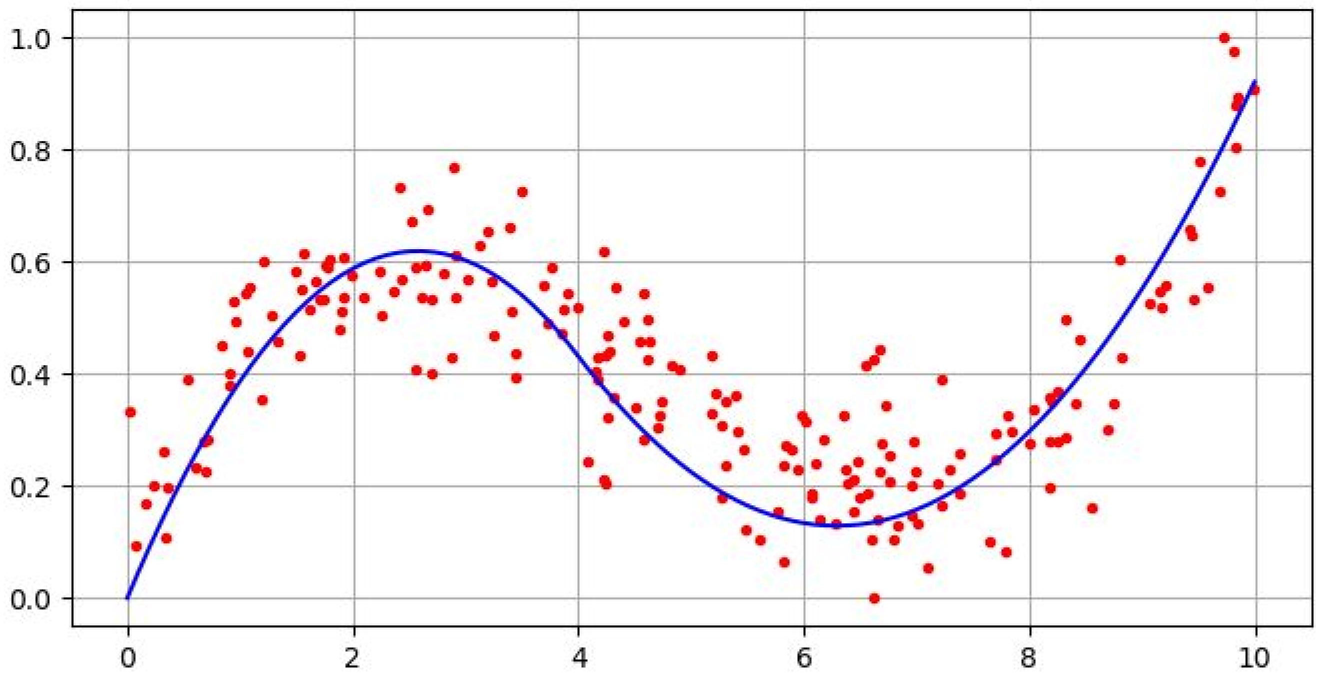
```
#The data to be fitted
println("Problem 2a.")
raw = readcsv("xy_data.csv");
x = raw[:,1];
y = raw[:,2];
T = length(x)

using PyPlot
figure(figsize=(10,4))
scatter(x,y, s=5)
grid("on")

#order of polynomial
k=3

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

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

@variable(m, a[1:k+1])
@constraint(m, a[k+1] == 0)
@objective(m, Min, sum((y-A*a).^2))
status = solve(m)
aopt = getvalue(a)

#plot the best-fit cubic
npts = 101
xfine = 0:0.1:10
ffine = ones(npts)

for j = 1:k
    ffine = [ffine.*xfine ones(npts)]
end

yfine = ffine * aopt
figure(figsize=(8,4))

plot(x, y, "r.", markersize=3)
plot(xfine, yfine, "b-")
grid()

k = 2 #polynomial order

#put data into groups of x<4 and x>=4
```

```
x1=[]
y1=[]
x2=[]
y2=[]

for i=1:length(x)
    if x[i] < 4
        push!(x1,x[i])
        push!(y1,y[i])
    else
        push!(x2,x[i])
        push!(y2,y[i])
    end
end

#create the A matrices

n1 = length(x1)
n2 = length(x2)
A1 = zeros(n1,k+1)
A2 = zeros(n2,k+1)

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

for i = 1:n2
    for j = 1:k+1
        A2[i,j] = x2[i]^(k+1-j)
    end
end

m2 = Model(solver = MosekSolver(LOG=0))
@variable(m2, p[1:k+1])
@constraint(m2, p[k+1] == 0)
@objective(m2, Min, sum((y1 - A1*p).^2))
status = solve(m2)
popt = getvalue(p)

m3 = Model(solver = MosekSolver(LOG=0))
@variable(m3, q[1:k+1])
@constraint(m3, 16*q[1]+4*q[2]+q[3] == (16*popt[1]+4*popt[2]+popt[3]))
@constraint(m3, 8*q[1]+q[2] == (8*popt[1]+popt[2]))
@objective(m3, Min, sum((y2 - A2*q).^2))
status = solve(m3)
qopt = getvalue(q)

xfine1 = 0:0.1:4
xfine2 = 4:0.1:10
ffine1 = ones(length(xfine1))
ffine2 = ones(length(xfine2))
```

```
for j = 1:k
    ffine1 = [ffine1.*xfine1 ones(length(xfine1))]
    ffine2 = [ffine2.*xfine2 ones(length(xfine2))]
end

yfine1 = ffine1 * popt
yfine2 = ffine2 * qopt
figure(figsize=(8,4))
plot([x1; x2], [y1; y2], "r.")
plot([xfine1; xfine2], [yfine1; yfine2], "b-")
grid()
```



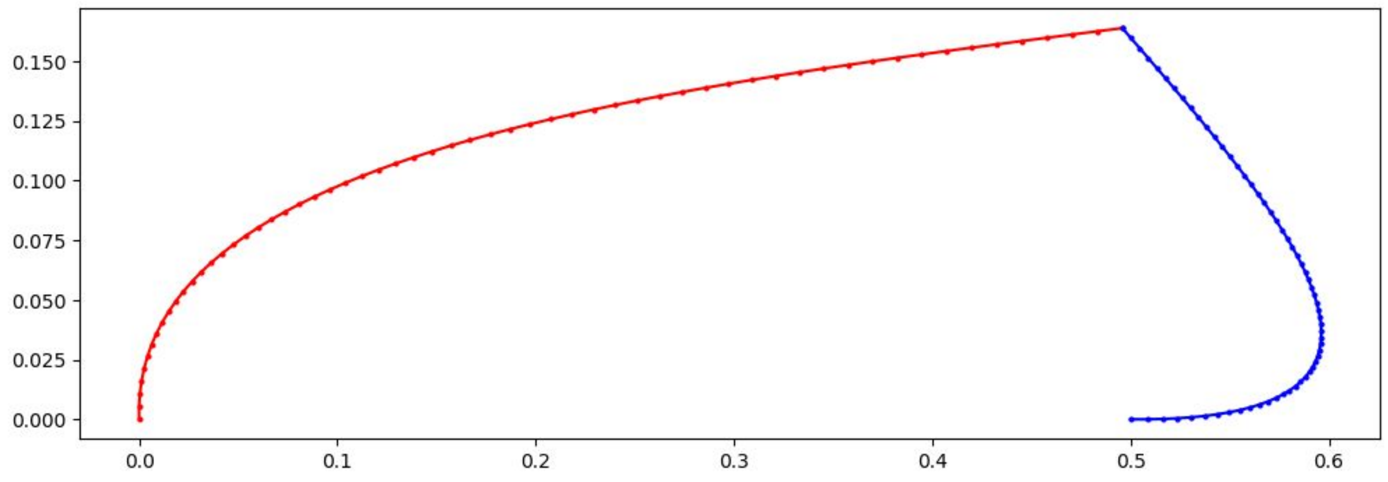




In [5]:

```
#Problem 3
using PyPlot, JuMP, Gurobi
T = 60
m1 = Model(solver = GurobiSolver(OutputFlag=0))
# length of time horizon
@variable(m1, xA1[1:2,1:T]) #position for Alice
@variable(m1, vA1[1:2,1:T]) #velocity for Alice
@variable(m1, uA1[1:2,1:T]) #thruster input for Alice
@variable(m1, xB1[1:2,1:T]) #position for Bob
@variable(m1, vB1[1:2,1:T]) #velocity for Bob
@variable(m1, uB1[1:2,1:T]) #thruster input for Bob
# constraint on start and end positions and velocities
@constraint(m1, vA1[:,1] .== [0;20]) #Alice goes north at 20mph
@constraint(m1, vB1[:,1] .== [30;0]) #Bob goes east at 30mph
@constraint(m1, xA1[:,1] .== [0;0]) # ALice starts at the origin
@constraint(m1, xB1[:,1] .== [0.5;0]) #Bob starts 0.5 miles east of Alice
@constraint(m1, xA1[:,60] .== xB1[:,60]) # at t = 60, Alice's and Bob's are at same location

# satisfy the dynamics
```

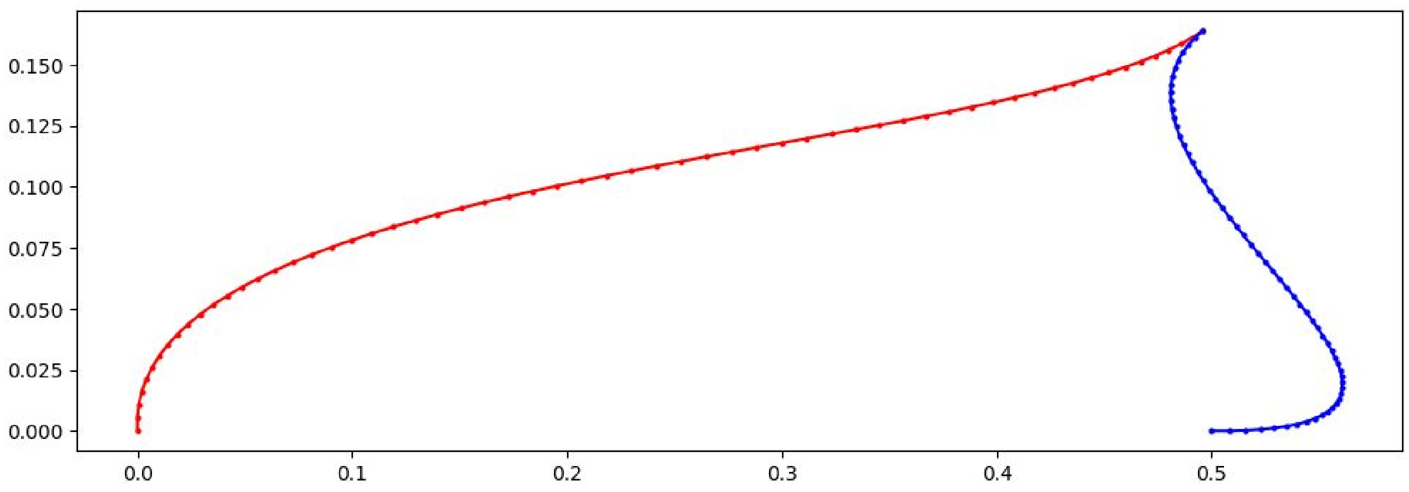
```
for t in 1:T-1
    @constraint(m1, xA1[:,t+1] .== xA1[:,t] + (1/3600)*vA1[:,t])
    @constraint(m1, xB1[:,t+1] .== xB1[:,t] + (1/3600)*vB1[:,t])
    @constraint(m1, vA1[:,t+1] .== vA1[:,t] + uA1[:,t])
    @constraint(m1, vB1[:,t+1] .== vB1[:,t] + uB1[:,t])
end

@objective(m1, Min, sum(uA1.^2)+sum(uB1.^2))
solve(m1)
uopt1 = getobjectivevalue(m1)
Alice1 = getvalue(xA1)
Bob1 = getvalue(xB1)
# Alice's maximum velocity
MaxVA1 = maximum(sqrt(getvalue(vA1[1,:]).^2 + getvalue(vA1[2,:]).^2))
figure(figsize=(12,4))
plot( Alice1[1,:][:], Alice1[2,:][:], "r.-", markersize=4 )
plot( Bob1[1,:][:], Bob1[2,:][:], "b.-", markersize=4 )

println("Optimal total energy is: ", uopt1)
println("Alice's location at t = 60 = (",Alice1[1,60],",",Alice1[2,60],")")
println("Bob's location at t = 60 = (", Bob1[1,60], ",", Bob1[2,60],")")
```



```
Optimal total energy is: 105.9307047910204
Alice's location at t = 60: (0.49583333333333333,0.16388888888888892)
Bob's location at t = 60: (0.49583333333333333,0.16388888888888892)
```

In [6]:

```julia
m2 = Model(solver = GurobiSolver(OutputFlag=0))
@variable(m2, xA2[1:2,1:T]) # resulting position for Alice
@variable(m2, vA2[1:2,1:T]) # resulting velocity for Alice
@variable(m2, uA2[1:2,1:T]) # thruster input for Alice
@variable(m2, xB2[1:2,1:T]) # resulting position for Bob
@variable(m2, vB2[1:2,1:T]) # resulting velocity for Bob
@variable(m2, uB2[1:2,1:T]) # thruster input for Bob

# constraint on start and end positions and velocities
@constraint(m2, vA2[:,1] .== [0;20]) # at t = 1, Alice goes N at 20mph
@constraint(m2, vB2[:,1] .== [30;0]) # at t = 1, Bob goes E at 30mph
@constraint(m2, xA2[:,1] .== [0;0]) # let Alice start at the origin
@constraint(m2, xB2[:,1] .== [0.5;0]) # Bob starts 0.5 miles E of Alice
@constraint(m2, xA2[:,60] .== xB2[:,60]) # at t = 60, Alice's and Bob's
# positions are the same
@constraint(m2, vA2[:,60] .== vB2[:,60]) # at t = 60, Alice's and Bob's velocities are the same

# satisfy the dynamics
for t in 1:T-1
    @constraint(m2, xA2[:,t+1] .== xA2[:,t] + (1/3600)*vA2[:,t])
    @constraint(m2, xB2[:,t+1] .== xB2[:,t] + (1/3600)*vB2[:,t])
    @constraint(m2, vA2[:,t+1] .== vA2[:,t] + uA2[:,t])
    @constraint(m2, vB2[:,t+1] .== vB2[:,t] + uB2[:,t])
end

@objective(m2, Min, sum(uA2.^2)+sum(uB2.^2))
solve(m2)
uopt2 = getobjectivevalue(m2)
Alice2 = getvalue(xA2)
Bob2 = getvalue(xB2)
VA = getvalue(vA2)
VB = getvalue(vB2)

figure(figsize=(12,4))
plot( Alice2[1,:][:], Alice2[2,:][:], "r.-", markersize=4 )
plot( Bob2[1,:][:], Bob2[2,:][:], "b.-", markersize=4 )

println("Optimal total energy is: ", uopt2)
println("Alice's location at t = 60 = (",Alice2[1,60],",",Alice2[2,60],")")
println("Bob's location at t = 60 = (", Bob2[1,60], ",", Bob2[2,60],")")
println("Alice's velocity at t = 60 = (", VA[1,60], ",", VA[1,60], ")")
println("Bob's velocity at t = 60 = (", VB[1,60], ",", VB[1,60], ")")
```



```
Optimal total energy is: 234.57042665108122
Alice's location at t = 60 = (0.4958333333333335,0.16388888888888883)
Bob's location at t = 60 = (0.4958333333333335,0.16388888888888883)
Alice's velocity at t = 60 = (15.000000000000009,15.000000000000009)
Bob's velocity at t = 60 = (15.000000000000009,15.000000000000009)
```