p2_hw5_me568_full

April 16, 2024

0.0.1 Part 2a - Install and Configure Julia Kernel

[ ]:
```julia
using Pkg
Pkg.status()
versioninfo()
```

```
Status `~/.julia/environments/v1.10/Project.toml`
  [336ed68f] CSV v0.10.14
  [a93c6f00] DataFrames v1.6.1
  [a98d9a8b] Interpolations v0.15.1
  [b6b21f68] Ipopt v1.6.2
  [4076af6c] JuMP v1.21.1
  [2fda8390] LsqFit v0.15.0
  [91a5bcdd] Plots v1.40.4
Julia Version 1.10.2
Commit bd47eca2c8a (2024-03-01 10:14 UTC)
Build Info:
  Official https://julialang.org/ release
Platform Info:
  OS: Linux (x86_64-linux-gnu)
  CPU: 8 × 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-15.0.7 (ORCJIT, tigerlake)
Threads: 1 default, 0 interactive, 1 GC (on 8 virtual cores)
Environment:
  LD_LIBRARY_PATH = /usr/lib/x86_64-linux-gnu/gazebo-11/plugins:/opt/ros/foxy/op
t/yaml_cpp_vendor/lib:/opt/ros/foxy/opt/rviz_ogre_vendor/lib:/opt/ros/foxy/lib/x
86_64-linux-gnu:/opt/ros/foxy/lib
  JULIA_NUM_THREADS =
```

0.0.2 Part 2b - Tire Force

[ ]:
```julia
using LsqFit
using CSV
using DataFrames
############## Here is a small example on how to use LsqFit ###############
# function model(input, par)
#     value = par[1] .* exp.( - input .* par[2])
#     return value
# end
# xdata = range(0, stop=10, length=20)
# ydata = model(xdata, [1.0 2.0]) + 0.01*randn(length(xdata))
# p0 = [0.5, 0.5]
# fit = curve_fit(model, xdata, ydata, p0)
# println("Your fitting value is: ", fit.param)
########## IMPORTANT COMMENT!!!: in function model(args), we add "." in front␣
 ↪of mathematical operators to allow broadcasting (similar to Matlab)␣
 ↪##########


function magicFormula(input, par)
    # par = [B, C]
    # input = xdata
    #TODO Fill in the magic formula equation here
    alpha = input[:,1]
    Fz = input[:,2]
    mu = input[:,3]
    B = par[1]
    C = par[2]
    # Fy = mu .* Fz .* sin(C .* atan((B./mu).* alpha))

    Fy = mu .* Fz .* sin.(C .* atan.((B./mu) .* alpha))

    return Fy
end

TireForceDataFrame = CSV.read("TireForce.csv", DataFrame) # Load data in␣
 ↪DataFrame mode, we recommend you to open csv to see the structure of data
TireForceMatrix = Matrix(TireForceDataFrame) # Change data format to matrix, it␣
 ↪is formatted in the form of [alpha Fz mu Fy], each one is a N x 1 array

# TODO prepare xdata and ydata from TireForceMatrix
xdata = TireForceMatrix[:, 1:3]
ydata = TireForceMatrix[:, end]

p0      = [1.7, 9.5]; # Initial Guess of [B, C]

#TODO Fill in function similar to the above example
fit     = curve_fit(magicFormula,xdata,ydata,p0)

B       = round(fit.param[1]; digits = 4)
C       = round(fit.param[2]; digits = 3)
println("B coefficient is: " ,B, "  C Coefficient is: ", C)
```

```
B coefficient is: 5.68  C Coefficient is: 1.817
```

0.0.3 Part 2c - Vehicle Bicycle Model

[ ]:
```
function VehicleDynamics(states, control)
    la = 1.56
    lb = 1.64
    m = 2020
    g = 9.81
    Izz = 4095
    h = 0.6
    mu = 0.8
    x = states[1]
    y = states[2]
    v = states[3]
    r = states[4]
      = states[5]
    ux = states[6]
     f = states[7]
    ax = control[1]
    d f = control[2]

    Fzf =  m *g * (lb/(la + lb)) - (m*h)/(la+lb) * ax #TODO Front axle load
    Fzr =  m *g * (la/(la + lb)) + (m*h)/(la+lb) * ax #TODO Rear axle load

     f =  f - atan((v + la * r)/ux) #TODO Front slip angle
     r = -atan((v-lb*r)/ux) #TODO Rear slip angle

    Fyf = MagicFormula( f, Fzf, mu) #TODO Front lateral force
    Fyr = MagicFormula( r, Fzr, mu) #TODO Rear lateral force

    dx          = ux * cos( ) - v * sin( )#TODO
    dy          = ux * sin( ) + v * cos( )#TODO
    dv          = ((Fyf + Fyr)/m) - ux * r#TODO
    dr          = (Fyf * la - Fyr * lb)/Izz#TODO
    d          =  r #TODO
    dux         = ax #TODO
    d          = d f #TODO
    dstates     = [dx dy dv dr d  dux d ]
    return dstates
end


function MagicFormula(alpha, Fz, mu)
    B =     5.68 #TODO Input Q2b value here
    C =    1.817 #TODO Input Q2b value here
    # Fy =  mu .* Fz .* sin.(C .* atan.((B./mu) .* alpha))  #TODO Lateral force␣
 ↪calculation
    Fy =  mu * Fz * sin(C * atan((B/mu) * alpha))
    return Fy
end


x0 = [-10.0 -5.0 0.5 0.1 0.1 10.0 0.1] # This is the initial state
ctrl = [1 0.1]  # One step control action
dstates = round.(VehicleDynamics(x0, ctrl); digits = 3) # Calculate states␣
 ↪derivative
println("The states derivative is: ", dstates)
```

```
The states derivative is: [9.9 1.496 -1.004 2.587 0.1 1.0 0.1]
```

[ ]:
```
transpose(dstates)
# dstates'
```

```
7×1 transpose(::Matrix{Float64}) with eltype Float64:
  9.9
  1.496
 -1.004
  2.587
  0.1
  1.0
  0.1
```

0.0.4 Part 2d - Vehicle Dynamics Propagation

[ ]:
```
# include("Q2c_VehicleDynamics.jl")

function Propagation(states, control,  T)
    #TODO Calculate states derivative using function
    #  VehicleDynamics(args).
    dstates =  VehicleDynamics(states, control)

    #TODO Calculate next Step
    # This is Explicit ForwardEuler
    statesNext = states + dstates .*  T
    return statesNext
end

#Testing purposes
x0 = [-10.0 -5.0 0.5 0.1 0.1 10.0 0.1] # This is the initial state
ctrl = [1 0.1]  # One step control action
 T = 0.01
# statesNextTemp = Propagation(x0,ctrl, T)
```

```
statesNextTemp = round.(Propagation(x0,ctrl, T); digits = 3)
println("The statesNextTemp  is: ", statesNextTemp)
```

```
The statesNextTemp  is: [-9.901 -4.985 0.49 0.126 0.101 10.01 0.101]
```

[ ]:
```
statesNext = StatesListFE02[1, :] .+ dstates .* dt1 #broadcast huh
```

```
7×7 Matrix{Float64}:
 -8.0  -10.0  -10.0  -10.0  -10.0  -9.8  -9.98
 -3.0   -5.0   -5.0   -5.0   -5.0  -4.8  -4.98
  2.0    0.0    0.0    0.0    0.0   0.2   0.02
  2.0    0.0    0.0    0.0    0.0   0.2   0.02
  2.0    0.0    0.0    0.0    0.0   0.2   0.02
 12.0   10.0   10.0   10.0   10.0  10.2  10.02
  2.0    0.0    0.0    0.0    0.0   0.2   0.02
```

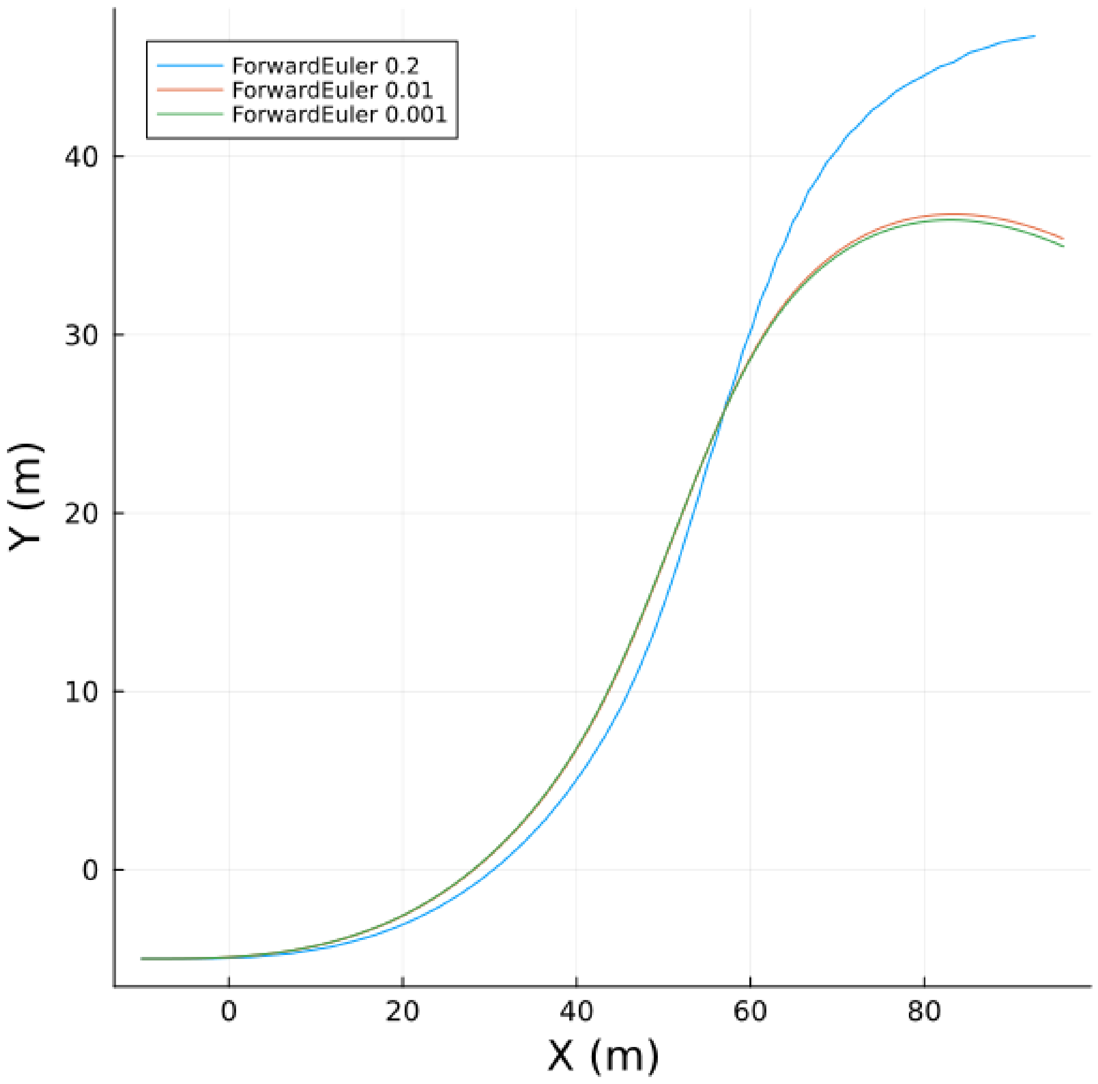
[ ]:
```
print(size(Propagation(reshape(StatesListFE02[1, :],(1,7)), control, dt1)))
```

```
(1, 7)
```

[ ]:
```
using Interpolations
using Plots
# include("Q2c_VehicleDynamics.jl")
# include("Q2d_StatesPropagator.jl")
x0 = [-10.0 -5.0 0.0 0.0 0.0 10.0 0.0]
ctrl = [1 0.1]
dstates = VehicleDynamics(x0, ctrl)


tc      = [0, 4, 8, 12]  # Key time step for control input
d fc    = [0, 0.02, -0.05, 0.02] # Key value for steering rate
axc     = [0, 1.0, -2.0, 1.0] # Key value for acceleration
dt1      = 0.2  # Simulation dt
t1       = 0:dt1:tc[end]
Interpolated f = interpolate((tc ,), d fc, Gridded(Constant{Next}())) #␣
 ↪Interpolations
Interpolateax = interpolate((tc ,), axc, Gridded(Constant{Next}()))
d f1    = Interpolated f.(t1) # Get interpolated steering rate signal
ax1     = Interpolateax.(t1) # Get interpolated acceleration signal

StatesListFE02 = zeros(size(t1, 1), size(x0, 2)) # Initialize states list for 0.
 ↪2 update time
StatesListFE02[1, :] = x0 # Initial point

control = zeros(2,1) # Init control input

for i = 1:size(StatesListFE02, 1) - 1
```

```
    # TODO calculate the next states
    control[1] = ax1[i]
    control[2] = d f1[i]
    StatesListFE02[i + 1, :] = Propagation(reshape(StatesListFE02[i, :],(1,7)),␣
  ↪control, dt1)
end

dt2     = 0.01 # Smaller time step
t2      = 0:dt2:tc[end]
d f2     = Interpolated f.(t2)
ax2      = Interpolateax.(t2)

StatesListFE001 = zeros(size(t2, 1), size(x0, 2)) # Initialize states list for␣
  ↪0.01 update time
StatesListFE001[1, :] = x0 # Initial point

for i = 1:size(StatesListFE001, 1) - 1

    # TODO calculate the next states
    control[1] = ax2[i]
    control[2] = d f2[i]
    StatesListFE001[i + 1, :] = Propagation(reshape(StatesListFE001[i, :
  ↪],(1,7)), control, dt2)
end


dt3     = 0.001 # Smaller time step
t3      = 0:dt3:tc[end]
d f3     = Interpolated f.(t3)
ax3      = Interpolateax.(t3)

StatesListFE0001 = zeros(size(t3, 1), size(x0, 2)) # Initialize states list for␣
  ↪0.001 update time
StatesListFE0001[1, :] = x0 # Initial point

for i = 1:size(StatesListFE0001, 1) - 1

    # TODO calculate the next states
    control[1] = ax3[i]
    control[2] = d f3[i]
    StatesListFE0001[i + 1, :] = Propagation(reshape(StatesListFE0001[i, :
  ↪],(1,7)), control, dt3)
end

p = plot(size = [600, 600])
```

```
plot!(p, StatesListFE02[:, 1], StatesListFE02[:, 2], label = "ForwardEuler " *␣
↪string(dt1), tickfontsize = 10, xlabel = "X (m)", ylabel = "Y␣
↪(m)",guidefont=15)
plot!(p, StatesListFE001[:, 1], StatesListFE001[:, 2], label = "ForwardEuler "␣
↪* string(dt2))
plot!(p, StatesListFE0001[:, 1], StatesListFE0001[:, 2], label = "ForwardEuler␣
↪" * string(dt3))
```

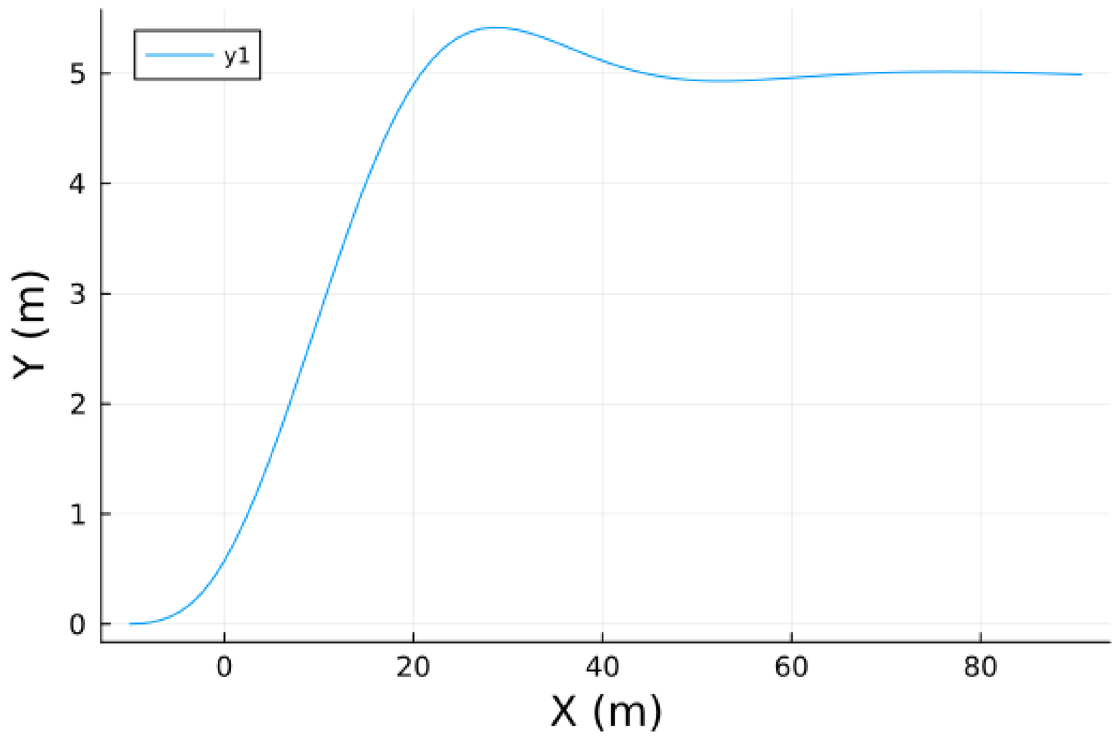


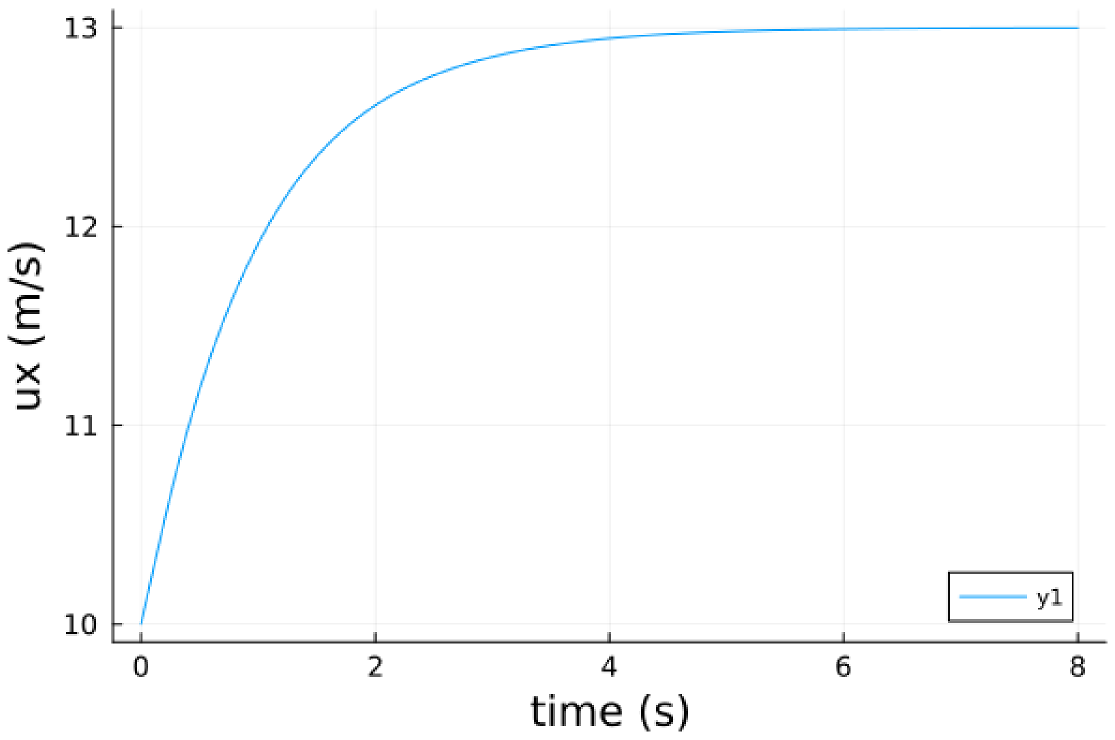
0.0.5 Part 2e - Optimal Control

Refer to : https://jump.dev/JuMP.jl/stable/tutorials/nonlinear/space_shuttle_reentry_trajectory/

[ ]:
```julia
using JuMP
using Ipopt
using Plots
# include("Q2c_VehicleDynamics.jl")

x0 = [-10.0, 0.0, 0.0, 0.0, 0.0, 10.0, 0.0] #TODO Initial Condition
XL = [-40, -20, -3, -pi/5,  -pi/2, 5.0, -pi/12] # States Lower Bound
XU = [300, 20, 3, pi/5, pi/2, 15.0, pi/12] #TODO States Upper Bound
CL = [-2.6, -0.1] #TODO Control Lower Bound
CU = [2.6, 0.1] #TODO Control Upper Bound

model = Model(optimizer_with_attributes(Ipopt.Optimizer)) # Initialize JuMP␣
 ↪model

numStates = 7 #TODO number of states
numControls = 2 #TODO number of control
PredictionHorizon = 8 #TODO Prediction Time
numColPoints = 81 #TODO
Δt = PredictionHorizon/(numColPoints - 1)# Time interval

@variables(model, begin
    # Set xst as a numColPoints x numStates matrix that is between the upper␣
 ↪and lower states bounds
    XL[i]   xst[j in 1:numColPoints, i in 1:numStates]    XU[i]
    #TODO Similarly, set u as a numColPoints x numControls matrix that is
    # between the upper and lower control bounds
    CL[i]   u[j in 1:numColPoints, i in 1:numControls]   CU[i]
end)

# Fix initial conditions
fix(xst[1, 1], x0[1]; force = true) # set the initial condition for x-position␣
 ↪value
#TODO Follow the same way, set the remaining initial conditions,
# set x0[2] to xst[1,2],... and so on.
fix(xst[1,2],x0[2];force = true)
fix(xst[1,3],x0[3];force = true)
fix(xst[1,4],x0[4];force= true)
fix(xst[1,5],x0[5];force= true)
fix(xst[1,6],x0[6];force= true)
fix(xst[1,7],x0[7];force= true)


# sa means steering angle, sr means steering rate
x = xst[:, 1]; y = xst[:, 2]; v = xst[:, 3]; r = xst[:, 4];   = xst[:, 5];
ux = xst[:, 6]; sa = xst[:, 7];
ax = u[:, 1]; # retract variable
sr = u[:, 2];
```

```
# xst = Matrix{Any}(undef, numColPoints, numStates)
# write the states derivative for all states & controls
 xst = Matrix{Any}(undef, numColPoints, numStates)
for i = 1:1:numColPoints
     xst[i, :] = @expression(model, VehicleDynamics(xst[i, :], u[i, :]))
    # xst[i, :] = @expression(model, VehicleDynamics(reshape(xst[i, :],(1,7)),
    #  reshape(u[i, :],(1,2))))
end

# add constraint to each state using backward Euler method
for j = 2:numColPoints
    for i = 1:numStates
        @constraint(model, xst[j, i] == xst[j - 1, i] +Δt *  xst[j, i])
    end
end


# TODO write the cost function for each term - Lane change
y_cost = @expression(model, sum((y[j] - 5)^2 * Δt for j= 1:1:numColPoints))␣
  ↪#global y position of C.G Cost
sr_cost = @expression(model, sum((sr[j])^2 * Δt for j= 1:1:numColPoints))
sa_cost = @expression(model, sum((sa[j])^2 * Δt for j= 1:1:numColPoints))
ux_cost = @expression(model, sum((ux[j] - 13)^2 * Δt for j= 1:1:numColPoints))
ax_cost = @expression( model, sum((ax[j])^2 * Δt for j=1:1:numColPoints)) # ax␣
  ↪cost


#TODO define cost weight
w_y  = 0.05 # change later for 2f
w_sr = 2.0
w_ax = 0.2
w_ux = 0.2
w_sa = 1.0

# Objective: Minimize cost function
@objective(model, Min, w_y * y_cost + w_sr * sr_cost + w_ax * ax_cost + w_ux *␣
  ↪ux_cost + w_sa * sa_cost) # objective value
optimize!(model) # optimize model
StatesHis = value.(model[:xst]) # retrieve data
if abs(objective_value(model) - 3.65) < 0.1 # check answer
    println("Congrats, your answer is correct")
else
    println("Something went wrong, please try again!")
end

println("Objective value model = ",objective_value(model))
println("Your y cost is:  ", round(value(y_cost); digits = 3))

#Plot
# plot(StatesHis[:, 1], StatesHis[:, 2], tickfontsize = 10, xlabel = "X (m)",␣
↪ylabel = "Y (m)",guidefont=15) # path plot
# plot(0:Δt:PredictionHorizon, StatesHis[:, 6], tickfontsize = 10, xlabel =␣
↪"time (s)", ylabel = "ux (m/s)",guidefont=15) # Speed plot

display(plot(StatesHis[:, 1], StatesHis[:, 2], tickfontsize = 10, xlabel = "X␣
↪(m)", ylabel = "Y (m)",guidefont=15)) # path plot
display(plot(0:Δt:PredictionHorizon, StatesHis[:, 6], tickfontsize = 10, xlabel␣
↪= "time (s)", ylabel = "ux (m/s)",guidefont=15)) # Speed plot)
```

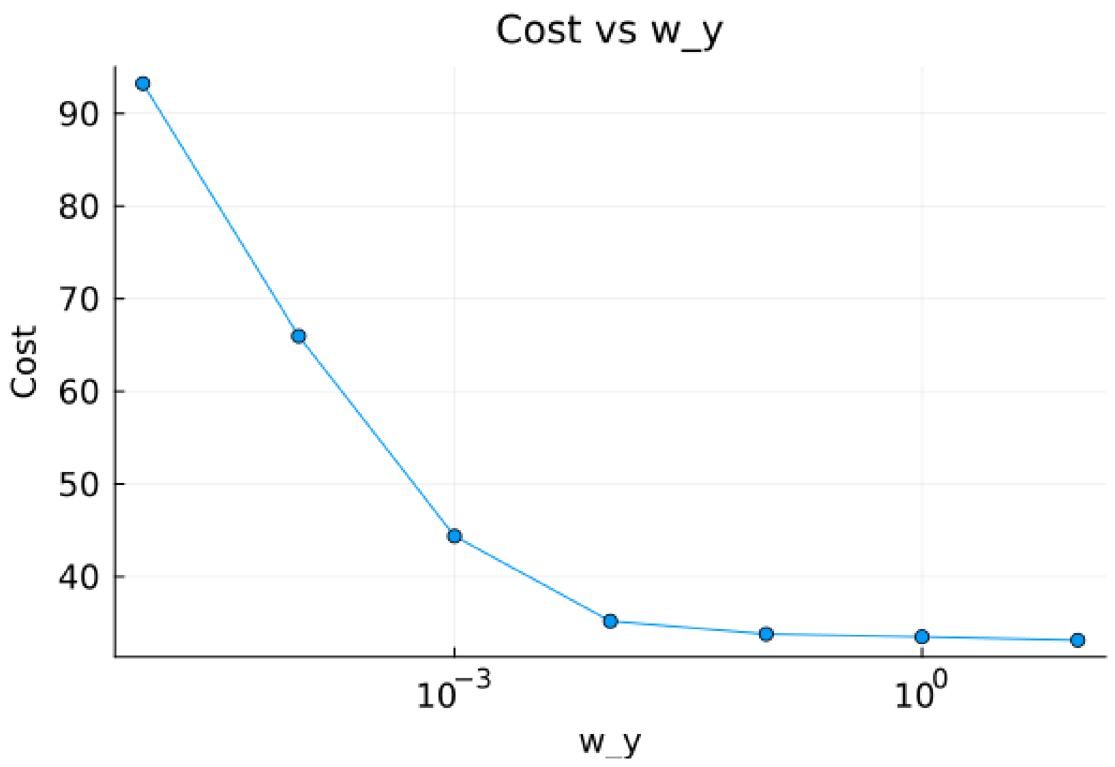




```
This is Ipopt version 3.14.14, running with linear solver MUMPS 5.6.2.

Number of nonzeros in equality constraint Jacobian…:     2473
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian…:     3602

Total number of variables…:      722
                     variables with only lower bounds:        0
                variables with lower and upper bounds:      722
                     variables with only upper bounds:        0
Total number of equality constraints…:      560
Total number of inequality constraints…:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  1.1142900e+02 9.50e+00 2.64e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  1.1109759e+02 8.23e+00 6.83e+00  -1.0 3.13e+01    -  1.67e-02 1.33e-01f  1
   2  1.1025958e+02 6.92e+00 5.68e+00  -1.0 2.83e+01    -  6.67e-02 1.60e-01f  1
   3  1.0819221e+02 5.04e+00 4.04e+00  -1.0 2.49e+01    -  9.68e-02 2.71e-01f  1
   4  1.0469717e+02 3.07e+00 2.31e+00  -1.0 1.94e+01    -  1.38e-01 3.90e-01f  1
   5  9.8916387e+01 1.03e+00 1.23e+00  -1.0 1.29e+01    -  2.15e-01 6.66e-01f  1
   6  9.1320242e+01 2.90e-03 7.91e-01  -1.0 5.54e+00    -  4.76e-01 1.00e+00f  1
   7  7.7137482e+01 1.85e-03 8.01e-01  -1.0 4.29e+00    -  5.18e-01 1.00e+00f  1
   8  3.2102711e+01 2.00e-02 3.97e-01  -1.0 2.02e+01    -  4.57e-01 1.00e+00f  1
   9  1.0894542e+01 1.70e-02 9.82e-02  -1.0 1.50e+01    -  6.62e-01 1.00e+00f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10  5.5093386e+00 4.06e-03 6.48e-02  -1.0 7.91e+00    -  1.00e+00 1.00e+00f  1
  11  4.2433091e+00 2.22e-03 1.18e-02  -1.7 3.76e+00    -  1.00e+00 1.00e+00f  1
  12  3.8205562e+00 2.76e-03 8.76e-03  -2.5 1.96e+00    -  9.80e-01 1.00e+00f  1
  13  3.6906527e+00 2.33e-03 3.35e-02  -3.8 5.20e-01    -  8.17e-01 1.00e+00h  1
  14  3.6620614e+00 6.66e-04 5.61e-04  -3.8 1.74e-01    -  1.00e+00 1.00e+00h  1
  15  3.6533282e+00 2.26e-04 2.77e-03  -5.7 7.89e-02    -  8.10e-01 9.78e-01h  1
  16  3.6519431e+00 2.36e-05 4.94e-04  -5.7 2.39e-02    -  9.47e-01 1.00e+00h  1
  17  3.6517338e+00 2.47e-06 5.25e-06  -5.7 8.66e-03    -  1.00e+00 1.00e+00h  1
  18  3.6516747e+00 4.23e-07 1.36e-05  -8.6 2.04e-03    -  9.85e-01 9.76e-01h  1
  19  3.6516712e+00 8.27e-08 3.93e-07  -8.6 1.09e-03    -  1.00e+00 1.00e+00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  20  3.6516708e+00 1.94e-08 9.49e-08  -8.6 5.38e-04    -  1.00e+00 1.00e+00h  1
  21  3.6516707e+00 4.52e-09 2.21e-08  -8.6 2.59e-04    -  1.00e+00 1.00e+00h  1
  22  3.6516707e+00 8.61e-10 4.21e-09  -8.6 1.13e-04    -  1.00e+00 1.00e+00h  1
  23  3.6516706e+00 1.12e-10 5.46e-10  -9.0 4.07e-05    -  1.00e+00 1.00e+00h  1

Number of Iterations…: 23

                                   (scaled)                 (unscaled)
Objective…:   3.6516706220464390e+00    3.6516706220464390e+00
Dual infeasibility…:   5.4583440954618428e-10    5.4583440954618428e-10
Constraint violation…:   1.1227421770065860e-10    1.1227421770065860e-10
Variable bound violation:   9.1102048405122815e-09    9.1102048405122815e-09
Complementarity…:   2.1361287151070329e-09    2.1361287151070329e-09
Overall NLP error…:   2.1361287151070329e-09    2.1361287151070329e-09


Number of objective function evaluations             = 24
Number of objective gradient evaluations             = 24
Number of equality constraint evaluations            = 24
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 24
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 23
Total seconds in IPOPT                               = 0.055

EXIT: Optimal Solution Found.
Congrats, your answer is correct
Objective value model = 3.651670622046439
Your y cost is:  33.942
```

0.0.6 Part 2f - Cost Weights

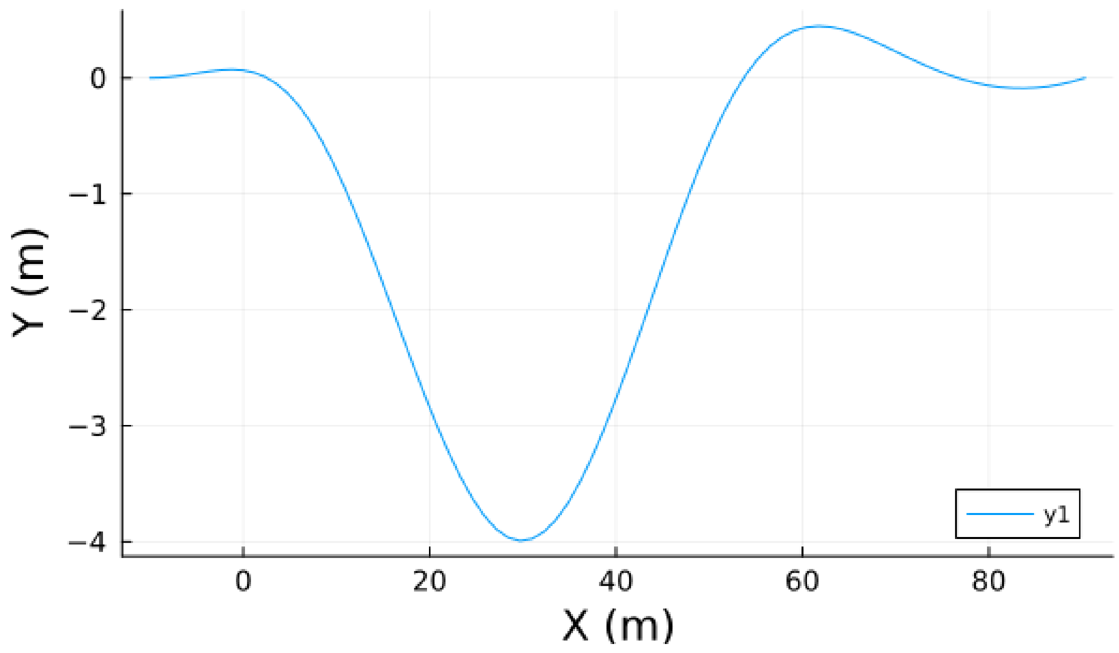
```
[ ]: # using Pkg
     # Pkg.add("Plots")
     using Plots

     w_y = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10]

     # TODO value of cost function
     cost = [93.24, 65.966, 44.373, 35.183, 33.798, 33.495, 33.135]

     # Using the `semilogx` function from Plots.jl to create a semilogarithmic plot
     # You can customize the plot using the `xlabel!`, `ylabel!`, and `title!`␣
      ↪functions
     # or by passing attributes directly within the `plot` function.
     plot(w_y, cost, xscale=:log10, markershape = :circle,
         xlabel="w_y", ylabel="Cost", title="Cost vs w_y", legend=false)

     # to customize the fontsize, you can use `fontsize()` function of the␣
      ↪underlying backend
     # For example, default GR backend:
     plot!(tickfontsize=12, labelfontsize=12, guidefontsize=12)
     # If the Plots backend you are using supports it, you can customize fonts␣
      ↪further.
```

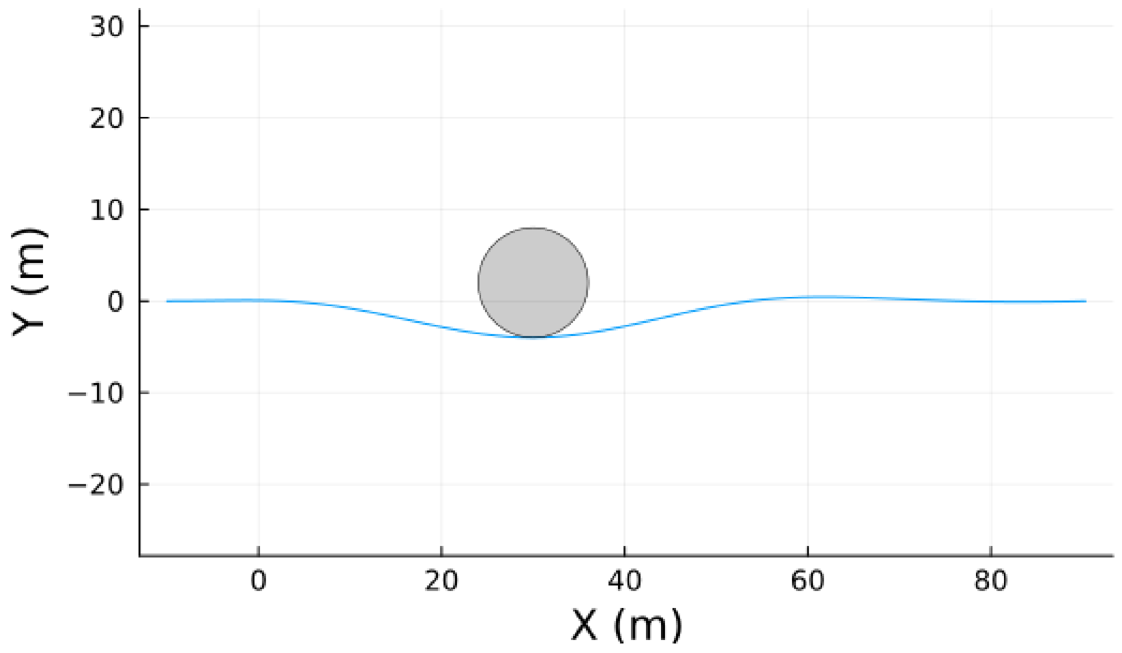


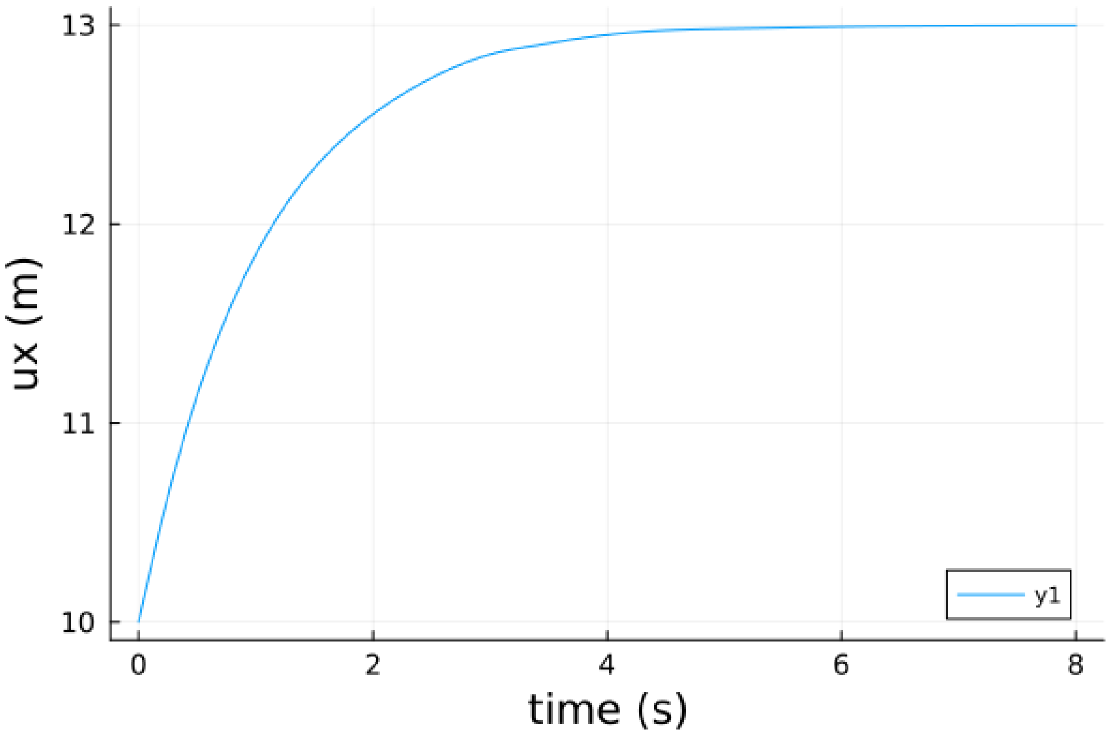
0.0.7 Part 2g - Obstacle Avoidance

[ ]:
```julia
using JuMP
using Ipopt
using Plots
# include("Q2c_VehicleDynamics.jl")

function circleShape(h,k,r)
     = LinRange(0, 2* , 500)
    h.+r*sin.( ), k.+r*cos.( )
end

x0 = [-10.0, 0.0, 0.0, 0.0, 0.0, 10.0, 0.0] #TODO Initial Condition
XL = [-40, -20, -3, -pi/5,  -pi/2, 5.0, -pi/12] # States Lower Bound
XU = [300, 20, 3, pi/5, pi/2, 15.0, pi/12] #TODO States Upper Bound
CL = [-2.6, -0.1] #TODO Control Lower Bound
CU = [2.6, 0.1] #TODO Control Upper Bound

model = Model(optimizer_with_attributes(Ipopt.Optimizer)) # Initialize JuMP␣
  ↪model

numStates = 7 #TODO number of states
numControls = 2 #TODO number of control
PredictionHorizon = 8 #TODO Prediction Time
numColPoints = 81 #TODO
Δt = PredictionHorizon/(numColPoints - 1)# Time interval

@variables(model, begin
    # Set xst as a numColPoints x numStates matrix that is
    # between the upper and lower states bounds
    XL[i]   xst[j in 1:numColPoints, i in 1:numStates]    XU[i]
    #TODO Set u as a numColPoints x numControls matrix that is between the
    #  upper and lower control bounds
    CL[i]   u[j in 1:numColPoints, i in 1:numControls]   CU[i]
end)


fix(xst[1, 1], x0[1]; force = true) # set the initial condition for x-position␣
  ↪value
#TODO Follow the same way, set the remaining initial conditions, set x0[2] to␣
  ↪xst[1,2],... and so on.
fix(xst[1,2],x0[2];force = true)
fix(xst[1,3],x0[3];force = true)
fix(xst[1,4],x0[4];force= true)
fix(xst[1,5],x0[5];force= true)
fix(xst[1,6],x0[6];force= true)
fix(xst[1,7],x0[7];force= true)


x = xst[:, 1]; y = xst[:, 2]; v = xst[:, 3]; r = xst[:, 4];   = xst[:, 5];
ux = xst[:, 6]; sa = xst[:, 7];
ax = u[:, 1]; # retract variable
sr = u[:, 2];

#  xst = Matrix{Any}(undef, numColPoints, numStates)
# write the states derivative for all states & controls
 xst = Matrix{Any}(undef, numColPoints, numStates)
for i = 1:1:numColPoints
    xst[i, :] = @expression(model, VehicleDynamics(xst[i, :], u[i, :]))
end

# add constraint to each state using backward Euler method
for j = 2:numColPoints
    for i = 1:numStates
        @constraint(model, xst[j, i] == xst[j - 1, i] +Δt *  xst[j, i])
    end
end


# TODO write the cost function for each term
y_cost = @expression(model, sum((y[j])^2 * Δt for j= 1:1:numColPoints)) #global␣
 ↪y position of C.G Cost
sr_cost = @expression(model, sum((sr[j])^2 * Δt for j= 1:1:numColPoints))
sa_cost = @expression(model, sum((sa[j])^2 * Δt for j= 1:1:numColPoints))
ux_cost = @expression(model, sum((ux[j] - 13)^2 * Δt for j= 1:1:numColPoints))
ax_cost = @expression( model, sum((ax[j])^2 * Δt for j=1:1:numColPoints)) # ax␣
 ↪cost



#TODO define cost weight
w_y  = 0.05 # change later for 2f
w_sr = 2.0
w_ax = 0.2
w_ux = 0.2
w_sa = 1.0


block_list = [30.0 2 6] # block_list = [obstacle_x_center, obstacle_y_center,␣
 ↪radius]

# TODO add obstacle avoidance constraint
```

```
obs_constraint = @constraint(model, [i = 1:numColPoints],
1 <=  ((x[i]-block_list[1])^2+(y[i]-block_list[2])^2)/(block_list[3])^2
)




@objective(model, Min, w_y * y_cost + w_sr * sr_cost + w_ax * ax_cost + w_ux *␣
 ↪ux_cost + w_sa * sa_cost) # objective value
optimize!(model) # optimize model
StatesHis = value.(model[:xst]) # retrieve data


println("Your y cost is:  ", round(value(y_cost); digits = 3))

p = plot(size=(600, 350))
display(plot!(p, StatesHis[:, 1], StatesHis[:, 2], tickfontsize = 10, xlabel =␣
 ↪"X (m)", ylabel = "Y (m)",guidefont=15))# path plot
display(plot!(p, circleShape(block_list[1], block_list[2], block_list[3]),␣
 ↪seriestype = [:shape,], ;w = 0.5, c=:black, linecolor = :black, legend =␣
 ↪false, fillalpha = 0.2, aspect_ratio=:equal))
display(plot(0:Δt:PredictionHorizon, StatesHis[:, 6], tickfontsize = 10, xlabel␣
 ↪= "time (s)", ylabel = "ux (m)",guidefont=15)) # Speed plot
```







```
This is Ipopt version 3.14.14, running with linear solver MUMPS 5.6.2.
```

```
Number of nonzeros in equality constraint Jacobian…:     2473
Number of nonzeros in inequality constraint Jacobian.:      320
Number of nonzeros in Lagrangian Hessian…:     3762

Total number of variables…:      722
                     variables with only lower bounds:        0
                variables with lower and upper bounds:      722
                     variables with only upper bounds:        0
Total number of equality constraints…:      560
Total number of inequality constraints…:       81
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:       81

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  1.0130400e+02 9.50e+00 7.65e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  1.0204698e+02 8.39e+00 5.41e+00  -1.0 4.79e+01    -  1.76e-02 1.16e-01H  1
   2  1.0188145e+02 7.57e+00 4.85e+00  -1.0 4.08e+01    -  6.42e-02 9.88e-02f  1
   3  1.0195396e+02 6.97e+00 4.46e+00  -1.0 2.86e+01    -  6.25e-02 7.92e-02h  2
   4  1.0200340e+02 6.96e+00 4.46e+00  -1.0 5.17e+02    -  1.39e-02 4.00e-04h  5
   5  1.0281406e+02 6.57e+00 4.21e+00  -1.0 4.85e+01    -  4.75e-02 5.63e-02h  2
   6  1.0485210e+02 5.25e+00 3.29e+00  -1.0 2.14e+01    -  5.92e-02 2.01e-01H  1
   7  1.0680535e+02 3.89e+00 5.31e+00  -1.0 1.24e+01    -  7.04e-02 2.60e-01H  1
   8  1.0782040e+02 3.15e+00 5.31e+00  -1.0 1.10e+01    -  6.68e-02 1.90e-01h  1
   9  1.0878320e+02 2.37e+00 3.87e+00  -1.0 1.04e+01    -  8.95e-02 2.48e-01f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10  1.0942251e+02 1.53e+00 2.31e+00  -1.0 9.65e+00    -  1.26e-01 3.54e-01f  1
  11  1.0885642e+02 6.66e-01 3.85e+00  -1.0 8.44e+00    -  2.02e-01 5.64e-01f  1
  12  1.0433388e+02 8.03e-02 4.65e+00  -1.0 5.84e+00    -  4.10e-01 8.79e-01f  1
  13  9.2102361e+01 2.81e-02 1.43e+00  -1.0 5.36e+00    -  5.11e-01 1.00e+00f  1
  14  6.3448871e+01 2.10e-01 1.03e+00  -1.0 1.50e+01    -  3.62e-01 1.00e+00f  1
  15  4.5590440e+01 4.65e-02 5.68e-01  -1.0 8.58e+00    -  5.22e-01 7.86e-01f  1
  16  3.9066811e+01 2.99e-02 1.09e+00  -1.0 1.12e+01    -  1.09e-01 3.62e-01f  1
  17  3.3439600e+01 1.79e-02 3.76e+00  -1.0 1.12e+01    -  6.46e-02 4.01e-01f  1
  18  2.5405825e+01 8.49e-03 1.47e+01  -1.0 9.70e+00    -  5.07e-02 9.06e-01f  1
  19  2.0638058e+01 2.57e-03 1.20e+01  -1.0 7.42e+00    -  6.55e-02 1.00e+00f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  20  1.7321446e+01 2.09e-03 7.31e+00  -1.0 1.13e+01    -  4.59e-01 4.00e-01f  1
  21  1.2523872e+01 4.57e-03 1.37e+01  -1.0 1.07e+01    -  4.95e-02 1.00e+00f  1
  22  1.0603519e+01 3.13e-03 8.06e+00  -1.0 1.15e+01    -  2.80e-01 3.79e-01f  1
  23  8.4492756e+00 2.14e-03 3.59e+00  -1.0 1.13e+01    -  1.44e-01 5.70e-01f  1
  24  8.2619953e+00 9.69e-04 1.84e+01  -1.0 3.64e+00    -  5.88e-02 1.00e+00f  1
  25  6.1076288e+00 1.28e-03 1.02e+01  -1.0 1.57e+01    -  4.31e-01 4.45e-01f  1
  26  5.7933335e+00 5.09e-04 6.29e+00  -1.0 3.70e+00    -  2.12e-01 1.00e+00f  1
  27  4.6635204e+00 9.67e-04 2.89e+00  -1.0 1.35e+01    -  5.41e-01 5.41e-01f  1
  28  4.9409531e+00 9.72e-05 1.90e+00  -1.0 3.78e-01    -  5.15e-01 1.00e+00f  1
  29  4.5356384e+00 4.20e-04 1.91e-01  -1.0 5.06e+00    -  1.00e+00 1.00e+00f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
```

```
  30  3.7970469e+00 4.89e-03 1.56e-01  -2.5 5.56e+00    -  8.97e-01 8.23e-01h  1
  31  3.4781707e+00 3.01e-03 1.33e-01  -2.5 1.96e+00    -  9.01e-01 1.00e+00h  1
  32  3.3747261e+00 1.93e-03 1.02e-03  -2.5 3.13e-01    -  1.00e+00 1.00e+00h  1
  33  3.3124500e+00 1.28e-03 3.04e-02  -3.8 2.51e-01    -  7.90e-01 1.00e+00h  1
  34  3.2934684e+00 3.17e-04 1.73e-04  -3.8 1.28e-01    -  1.00e+00 1.00e+00h  1
  35  3.2856980e+00 1.05e-04 7.06e-04  -5.7 6.01e-02    -  9.43e-01 1.00e+00h  1
  36  3.2848904e+00 7.96e-06 6.69e-06  -5.7 1.71e-02    -  1.00e+00 1.00e+00h  1
  37  3.2848271e+00 7.67e-07 5.38e-07  -5.7 4.61e-03    -  1.00e+00 1.00e+00h  1
  38  3.2847616e+00 1.05e-07 4.82e-06  -8.6 2.44e-03    -  9.93e-01 1.00e+00h  1
  39  3.2847602e+00 4.75e-09 1.13e-09  -8.6 3.99e-04    -  1.00e+00 1.00e+00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  40  3.2847602e+00 1.39e-11 3.86e-12  -8.6 2.36e-05    -  1.00e+00 1.00e+00h  1

Number of Iterations…: 40

                                   (scaled)                 (unscaled)
Objective…:   3.2847602184500948e+00    3.2847602184500948e+00
Dual infeasibility…:   3.8614831090414654e-12    3.8614831090414654e-12
Constraint violation…:   1.3932396902838207e-11    1.3932396902838207e-11
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity…:   2.7585141641059175e-09    2.7585141641059175e-09
Overall NLP error…:   2.7585141641059175e-09    2.7585141641059175e-09


Number of objective function evaluations             = 55
Number of objective gradient evaluations             = 41
Number of equality constraint evaluations            = 55
Number of inequality constraint evaluations          = 55
Number of equality constraint Jacobian evaluations   = 41
Number of inequality constraint Jacobian evaluations = 41
Number of Lagrangian Hessian evaluations             = 40
Total seconds in IPOPT                               = 0.114

EXIT: Optimal Solution Found.
Your y cost is:  25.81
```