Team Homework 2 - MATH 420

James Boggs and Camilo Velez R

NYC Covid Data

In [1]:
```
# activate env and instantiate pkgs
using Pkg
Pkg.activate("../p1")
Pkg.instantiate()
```

```
  Activating project at `~/m420p1/p1`
```

In [2]:
```
using CSV, DataFrames, Dates, LaTeXStrings, Plots, Distances, OrderedCollect
```

In [3]:
```
df = CSV.read("../resources/data.csv", DataFrame) # entire df

v_infected = values(df[1, 13:end]) # vector of infected numbers
infected_dates = names(df[1, 13:end]) # vector of infected_dates
@assert length(v_infected) == length(infected_dates)
```

Exercise 1

1)

In [4]:
```
yd = values(df[2, 13:end]);
```

In [5]:
```
v(t::Int)::Int = v_infected[t]
y(t::Int)::Int = yd[t];
```

In [6]:
```
findfirst([x>=5 for x in v_infected])
```

Out[6]:
```
46
```

In [7]:
```
y_deaths = values(df[2, 13+46-1:end]); # vector of deaths
```

$T_0 = 46$

In [8]:
```
function I(t::Int, τ::Int=7)::Real
    t0 = 46
    return v(t+t0+τ) - v(t+t0-τ)
end
```

Out[8]:
```
I (generic function with 2 methods)
```

In [9]:
```julia
popu=df.Population[1];
tmax = 119;
```

In [10]:
```julia
# df = CSV.read("../resources/data.csv", DataFrame) # entire df

# read the values of infections and deaths from the table
infected = values(df[1, 13:end]) # vector of infected numbers
deaths = values(df[2, 13:end]) # vector of death numbers
infected_dates = names(df[1, 13:end]) # vector of infected_dates
population = df.Population[1];
@assert length(infected) == length(deaths)
# parameters
Vmin = 5
τ0 = 7
λ = 1

t0 = findfirst([x>=Vmin for x in infected])
tmax = 119
# define V, Y, and I for the range of times we are interested in
V(t) = infected[t + t0]
Y(t) = deaths[t + t0]
I(t) = infected[t + t0 + τ0] - infected[t + t0 - τ0]
```

Out[10]:
```
I (generic function with 3 methods)
```

In [11]:
```julia
# function to find the optimal gamma for a given R_sim and p
function optGamma(R_sim, p)
    if p == 2
        s1(t) = Y(t)*R_sim[t+1]
        s2(t) = abs(R_sim[t+1])^2
        gamma_c = sum(s1, 0:tmax)/sum(s2, 0:tmax)
        if gamma_c < 0
            return 0
        elseif gamma_c > 1
            return 1
        else
            return gamma_c
        end
    elseif p == 1
        # min = Inf
        # minimizer = -1
        md = Dict()
        for k in 0:tmax
            (R_sim[k+1]==0) && continue
            r = Y(k)/R_sim[k+1]
            if r >= 0 && r <= 1

                s3(t) = abs(Y(t) - r*R_sim[t+1])
                #f = sum(s3, 0:tmax)
                md[r] = Cityblock()(Y.(0:tmax), r*R_sim) #sum(s3, 0:tmax)
                # if f < min
                #     min = f
                #     minimizer = r
                # end
            end
```

```
        end


        return argmin(md)
    else
        model = Model()
        set_optimizer(model,GLPK.Optimizer)

        @variable(model, x1>= 0)
        @variable(model, 1>=x2>=0)

        for t = 0:tmax
            @constraint(model, (-x1) - R_sim[t+1]*x2 <= -Y(t))
            @constraint(model, (-x1) + R_sim[t+1]*x2 <= Y(t))
        end

        @objective(model, Min, x1)
        optimize!(model)

        return value.(x2)
    end
end
```

Out[11]: optGamma (generic function with 1 method)

In [12]:

```
# Euler scheme
function euler(alpha, beta, N)::NTuple{3, Vector{Float64}}
  h = 0.01
  n = 2

  S_sim = Vector{Float64}(undef, 120)
  I_sim = Vector{Float64}(undef, 120)
  R_sim = Vector{Float64}(undef, 120)

  s = N
  i = I(0)
  r = 0

  t = 0

  S_sim[1] = s
  I_sim[1] = i
  R_sim[1] = r
  while n <= 120
    ds = -beta*s*(i/N)
    di = beta*s*(i/N) - alpha*i
    dr = alpha*i

    s += h*ds
    i += h*di
    r += h*dr
    t += h
    rt = round(t, digits=2);
    if isinteger(rt)
      S_sim[n] = s
      I_sim[n] = i
```

```
      R_sim[n] = r
      n += 1
    end
  end

  return S_sim, I_sim, R_sim
end
```

Out[12]: euler (generic function with 1 method)

In [13]:
```
Y1 = first(y_deaths, 120);
```

In [14]:
```
function J1(γ, Is, Rs, p)::Real
    λ = 1
    f1(t) = abs(I(t) - Is[t+1])^p
    f2(t) = abs(y_deaths[t+1] - γ*Rs[t+1])^p
    return sum(f1, 0:119) + λ*sum(f2, 0:119)
end
```

Out[14]: J1 (generic function with 1 method)

$p = 1$

In [47]:
```
Jd = OrderedDict()
```

Out[47]: OrderedDict{Any, Any}()

In [48]:
```
for α in ProgressBar(0.05:0.01:0.2)
    for r0 in 1.5:0.1:1.9
        for nn in 2:0.5:10
            β = r0*α
            N = popu*nn/100
            Ss, Is, Rs = euler(α, β, N)
            #ds = OrderedDict()
            #for γ in 0:0.0001:1
            #    ds[γ] = Cityblock()(Y1, γ*Rs)
            #end
            #γ̂ = argmin(ds)
            γ̂  = optGamma(Rs, 1)
            Jd[(α, β, r0, N, γ̂)] = J1(γ̂, Is, Rs, 1)
        end
    end
end
```

```
0.0%|                                              | 0/16 [00:00<00:-1, -0
s/it]
6.2%|███                                        | 1/16 [00:02<Inf:Inf, InfG
s/it]
12.5%|██████                                       | 2/16 [00:06<01:21, 6
s/it]
18.8%|█████████                                    | 3/16 [00:10<01:05, 5
s/it]
25.0%|████████████                                 | 4/16 [00:15<01:01, 5
s/it]
31.2%|███████████████                              | 5/16 [00:21<00:57, 5
s/it]
37.5%|██████████████████                           | 6/16 [00:27<00:54, 5
s/it]
43.8%|█████████████████████                        | 7/16 [00:33<00:50, 6
s/it]
50.0%|████████████████████████                     | 8/16 [00:39<00:45, 6
s/it]
56.2%|███████████████████████████                  | 9/16 [00:46<00:40, 6
s/it]
62.5%|█████████████████████████████               | 10/16 [00:52<00:35, 6
s/it]
68.8%|████████████████████████████████            | 11/16 [00:58<00:29, 6
s/it]
75.0%|███████████████████████████████████         | 12/16 [01:05<00:24, 6
s/it]
81.2%|██████████████████████████████████████      | 13/16 [01:11<00:18, 6
s/it]
87.5%|█████████████████████████████████████████   | 14/16 [01:18<00:12, 6
s/it]
93.8%|████████████████████████████████████████████| 15/16 [01:24<00:06, 6
s/it]
100.0%|█████████████████████████████████████████████| 16/16 [01:30<00:00, 6
s/it]
100.0%|█████████████████████████████████████████████| 16/16 [01:30<00:00, 6
s/it]
```

In [49]:
```
α̂, β̂, r0̂, N̂, γ̂ = argmin(Jd)
J_min = Jd[argmin(Jd)];
```

In [50]:
```
display(L"\text{For}  p=1")
@show α̂, β̂, r0̂, N̂, γ̂;
display(L"J_{min}\approx %$(round(J_min, digits=3))")
```

$\text{For} \quad p = 1$

```
(α̂, β̂, r0̂, N̂, γ̂) = (0.17, 0.323, 1.9, 48861.18, 0.0789682964249939)
```

$J_{min} \approx 210762.063$

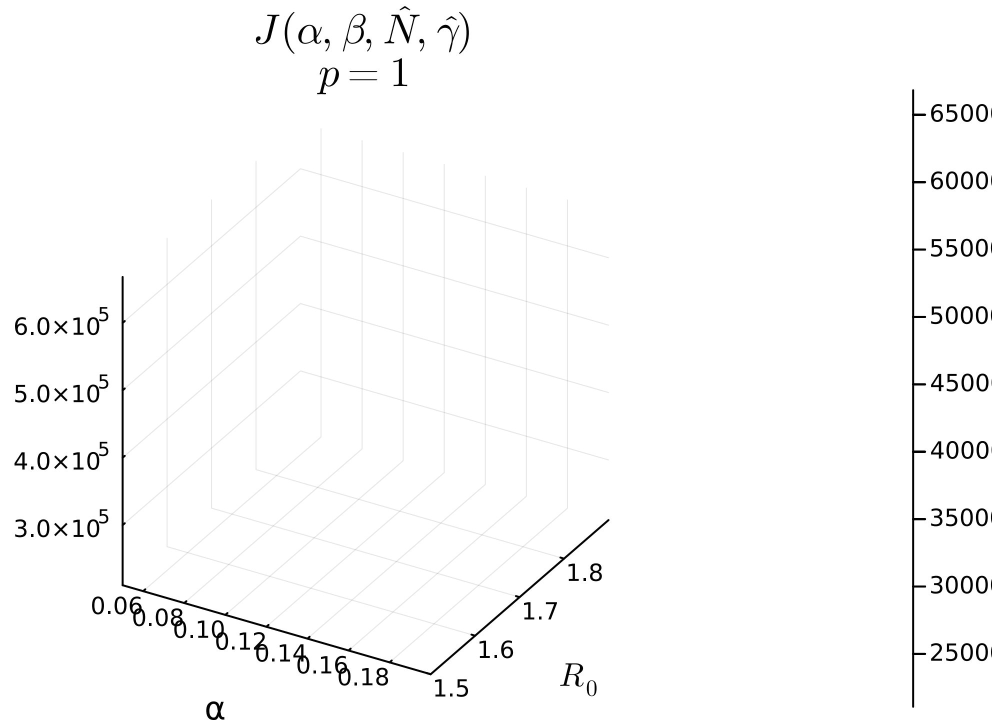
2)

In [51]:
```
function Jplot(α, β, p)
    S, I, R = euler(α, β, N̂)
```

```julia
    return J1(γ̂, I, R, p)
end
```

Out[51]: Jplot (generic function with 1 method)

In [55]:
```julia
display(surface(0.05:0.01:0.2, 1.5:0.01:1.9, (x,y) -> Jplot(x, x*y, 1), titl
png("p1surface_exercise1");
```

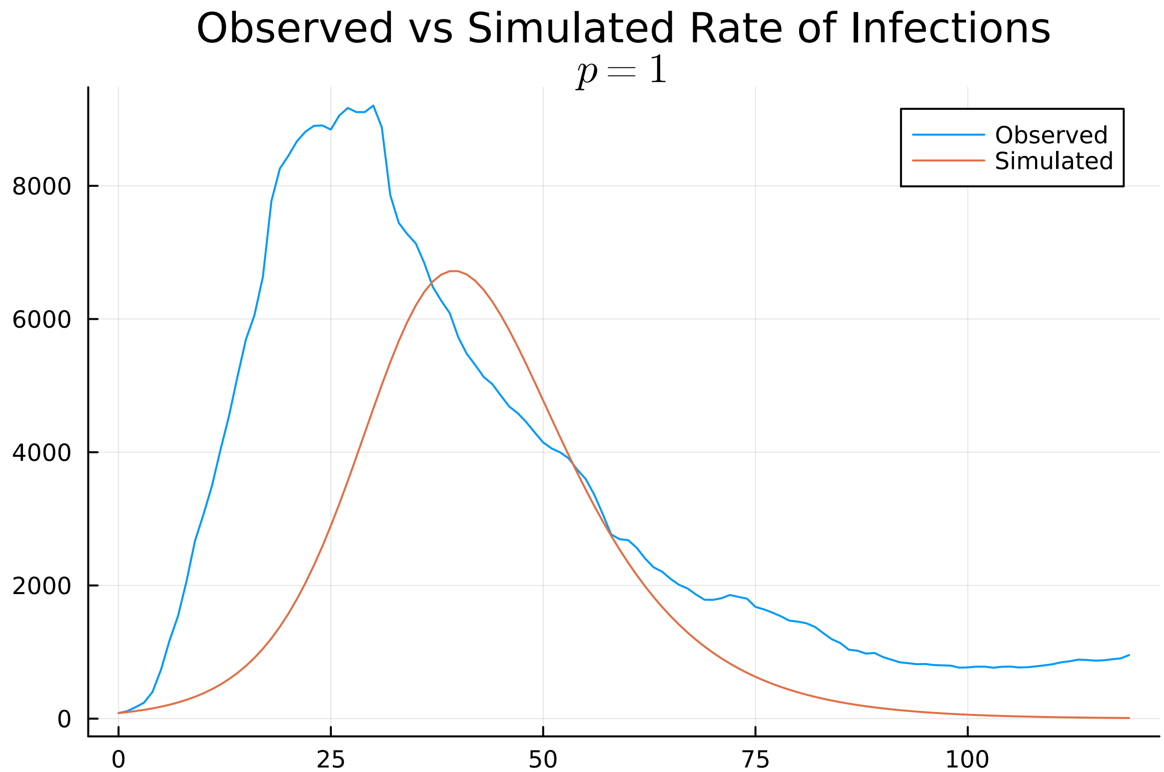


3)

In [21]:
```julia
Ss, Is, Rs = euler(α̂, β̂, N̂);
```

In [22]:
```julia
plot(0:119, I, labels="Observed")
plot!(0:119, Is, label="Simulated")
plot!(title = "Observed vs Simulated Rate of Infections" * "\n" * L"p=1")
```

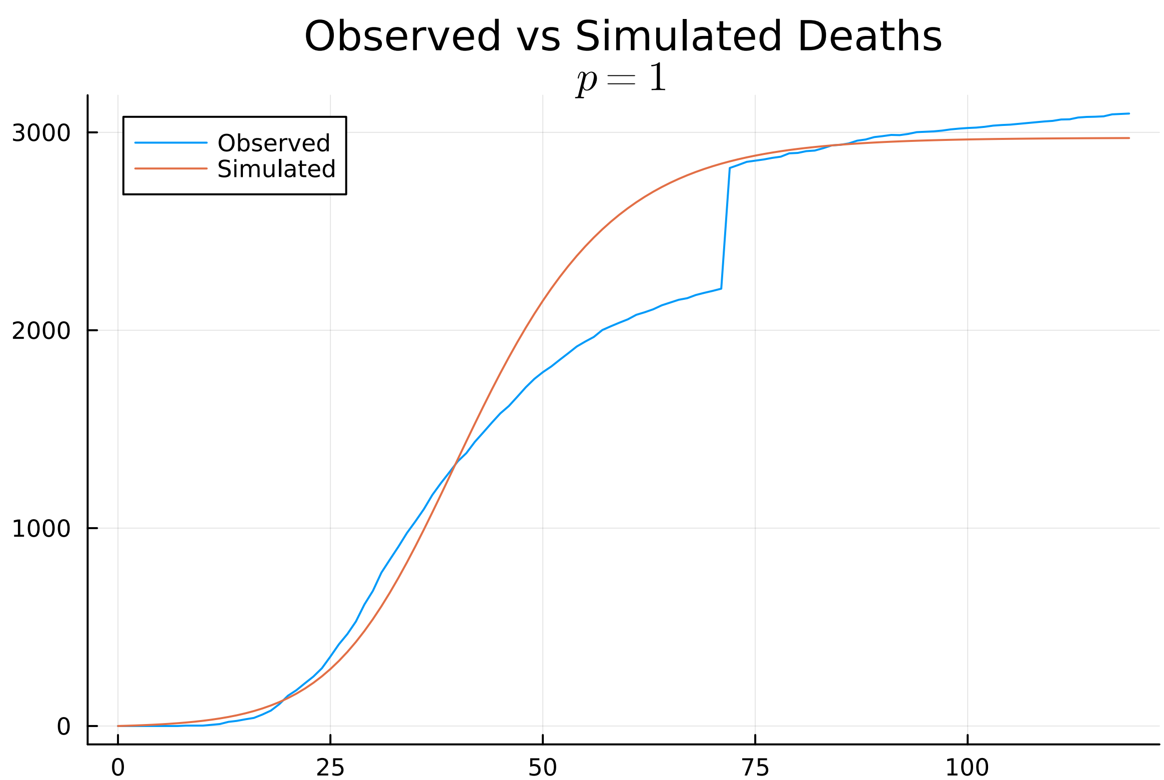
Out[22]:



In [23]:

```
plot(0:119, Y1, labels="Observed")
plot!(0:119, γ̂*Rs, label="Simulated")
plot!(title = "Observed vs Simulated Deaths" * "\n" * L"p=1")
```

Out[23]:



$p = 2$

In [56]:
```
Jd = OrderedDict()
for α in 0.05:0.01:0.2
    for r0 in 1.5:0.1:1.9
        for nn in 2:0.5:10
            β = r0*α
            N = popu*nn/100
            Ss, Is, Rs = euler(α, β, N)
            #ds = OrderedDict()
            #for γ in 0:0.0001:1
            #    ds[γ] = Euclidean()(Y1, γ*Rs)
            #end
            #γ̂ = argmin(ds)
            γ̂  = optGamma(Rs, 2)
            Jd[(α, β, r0, N, γ̂)] = J1(γ̂, Is, Rs, 2)
        end
    end
end
```

In [57]:
```
α̂, β̂, r0̂, N̂, γ̂ = argmin(Jd)
J_min = Jd[argmin(Jd)];
```

In [58]:
```
display(L"\text{For}  p=2")
@show α̂, β̂, r0̂, N̂, γ̂;
display(L"J_{min}\approx %$(round(J_min, digits=3))")
```

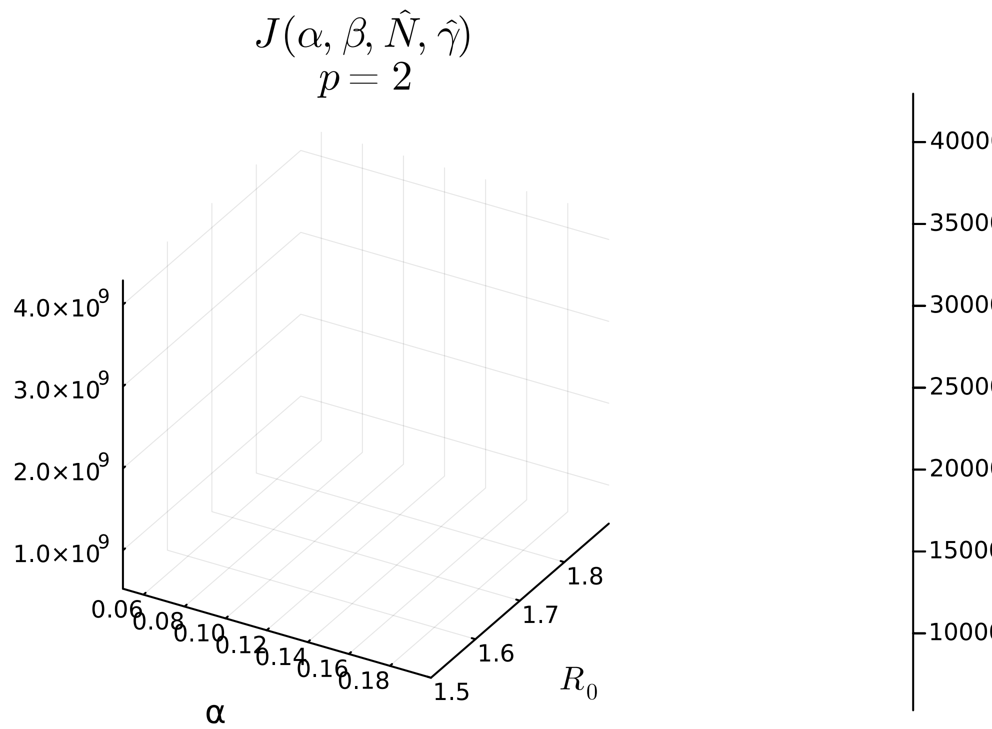
For $\quad p = 2$

```
(α̂, β̂, r0̂, N̂, γ̂) = (0.2, 0.38, 1.9, 65148.24, 0.053995703004223516)
```

$J_{min} \approx 5.29978911123e8$

2)

In [59]:
```
display(surface(0.05:0.01:0.2, 1.5:0.01:1.9, (x,y) -> Jplot(x, x*y, 2), titl
png("p2surface_exercise1");
```

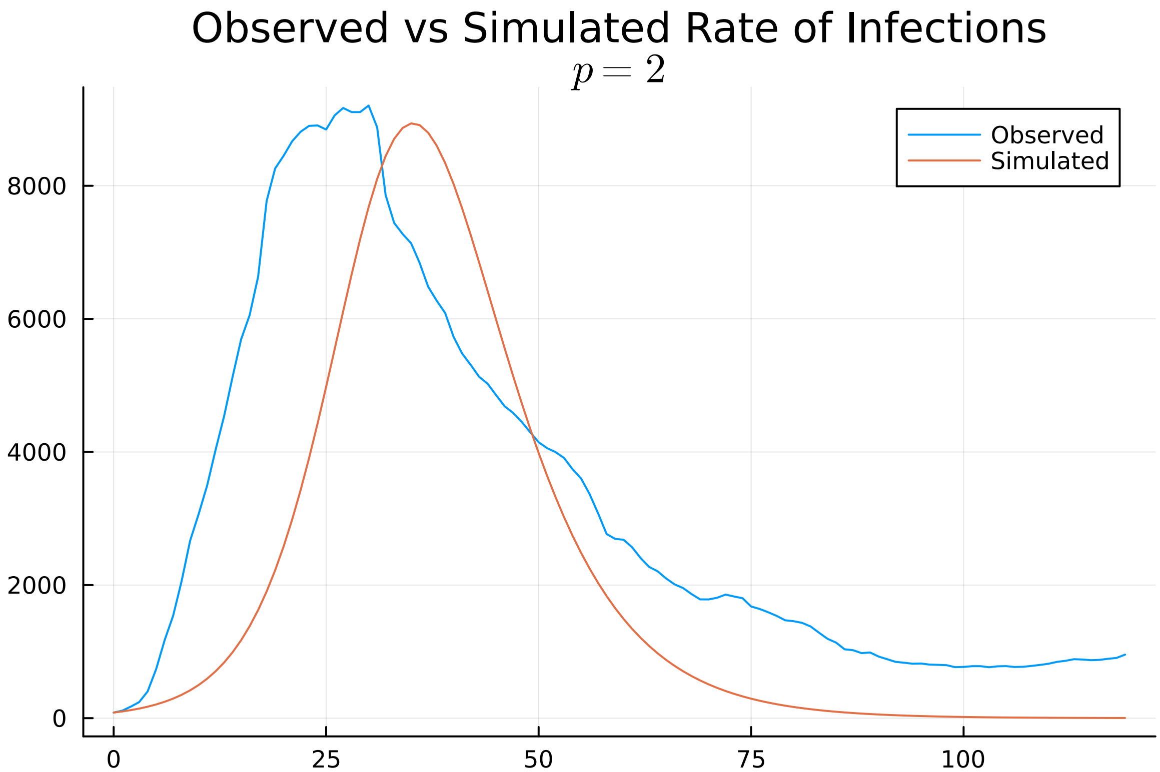


3)

In [60]:
```
Ss, Is, Rs = euler(α̂, β̂, N̂);
```

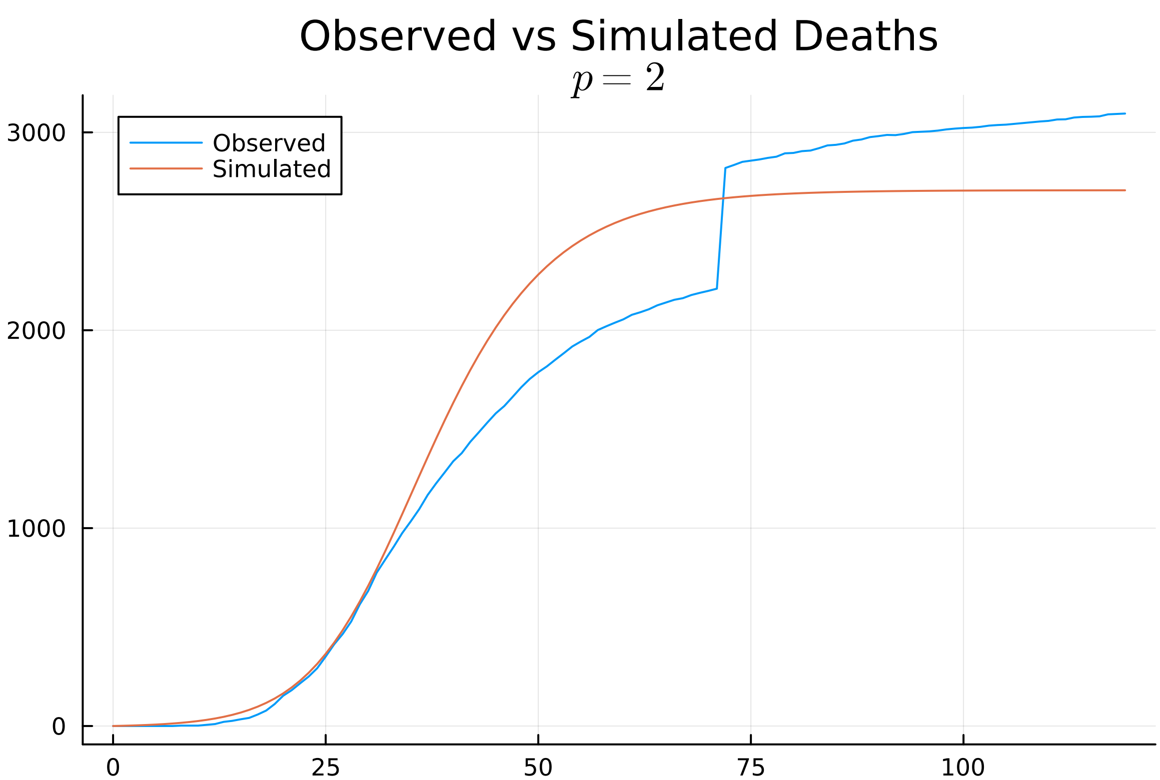
In [61]:
```
plot(0:119, I, labels="Observed")
plot!(0:119, Is, label="Simulated")
plot!(title = "Observed vs Simulated Rate of Infections" * "\n" * L"p=2")
```

Out[61]:



In [62]:
```
plot(0:119, Y1, labels="Observed")
plot!(0:119, γ̂*Rs, label="Simulated")
plot!(title = "Observed vs Simulated Deaths" * "\n" * L"p=2")
```

Out[62]:



$p = \infty$

In [63]:
```julia
function J2(γ, Is, Rs, p)::Real
    λ = 1
    f1(t) = abs(I(t) - Is[t+1])
    f2(t) = abs(y_deaths[t+1] - γ*Rs[t+1])
    return maximum(f1, 0:119) + λ*maximum(f2, 0:119)
end
```

Out[63]: J2 (generic function with 1 method)

In [64]:
```julia
Jd = OrderedDict()
for α in 0.05:0.01:0.2
    for r0 in 1.5:0.1:1.9
        for nn in 2:0.5:10
            β = r0*α
            N = popu*nn/100
            Ss, Is, Rs = euler(α, β, N)
            ds = OrderedDict()
            #for γ in 0:0.01:1
            #    ds[γ] = maximum(abs.(Y1 - γ*Rs))
            #end
            #γ̂ = argmin(ds)
            γ̂  = optGamma(Rs, 3)
            Jd[(α, β, r0, N, γ̂)] = J1(γ̂, Is, Rs, 2)
        end
    end
end
```

In [65]:
```julia
α̂, β̂, r0̂, N̂, γ̂ = argmin(Jd)
J_min = Jd[argmin(Jd)];
```

In [66]:
```julia
display(L"\text{For}  p=∞")
@show α̂, β̂, r0̂, N̂, γ̂;
display(L"J_{min}\approx %$(round(J_min, digits=3))")
```

$$\text{For}\quad p = \infty$$

(α̂, β̂, r0̂, N̂, γ̂) = (0.2, 0.38, 1.9, 65148.24, 0.05269036383890261)

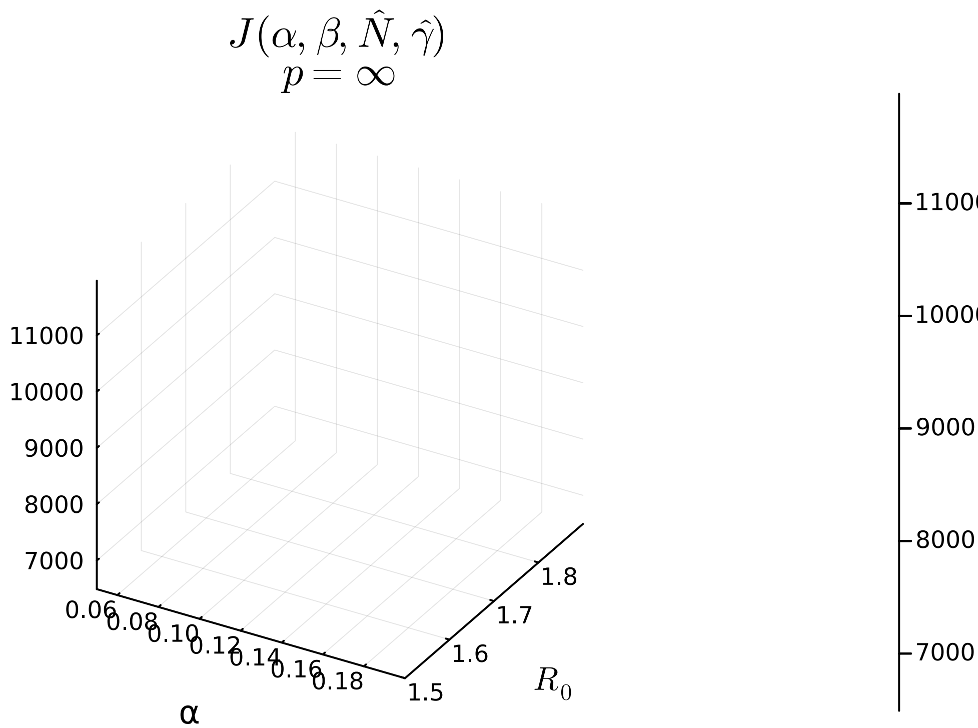
$$J_{min} \approx 5.30297452273e8$$

In [67]:
```julia
function Jplot2(α, β, p)
    S, I, R = euler(α, β, N̂)
    return J2(γ̂, I, R, p)
end
```

Out[67]: Jplot2 (generic function with 1 method)

2)

In [68]:
```julia
display(surface(0.05:0.01:0.2, 1.5:0.01:1.9, (x,y) -> Jplot2(x, x*y, 2), tit
png("pinfsurface_exercise1");
```

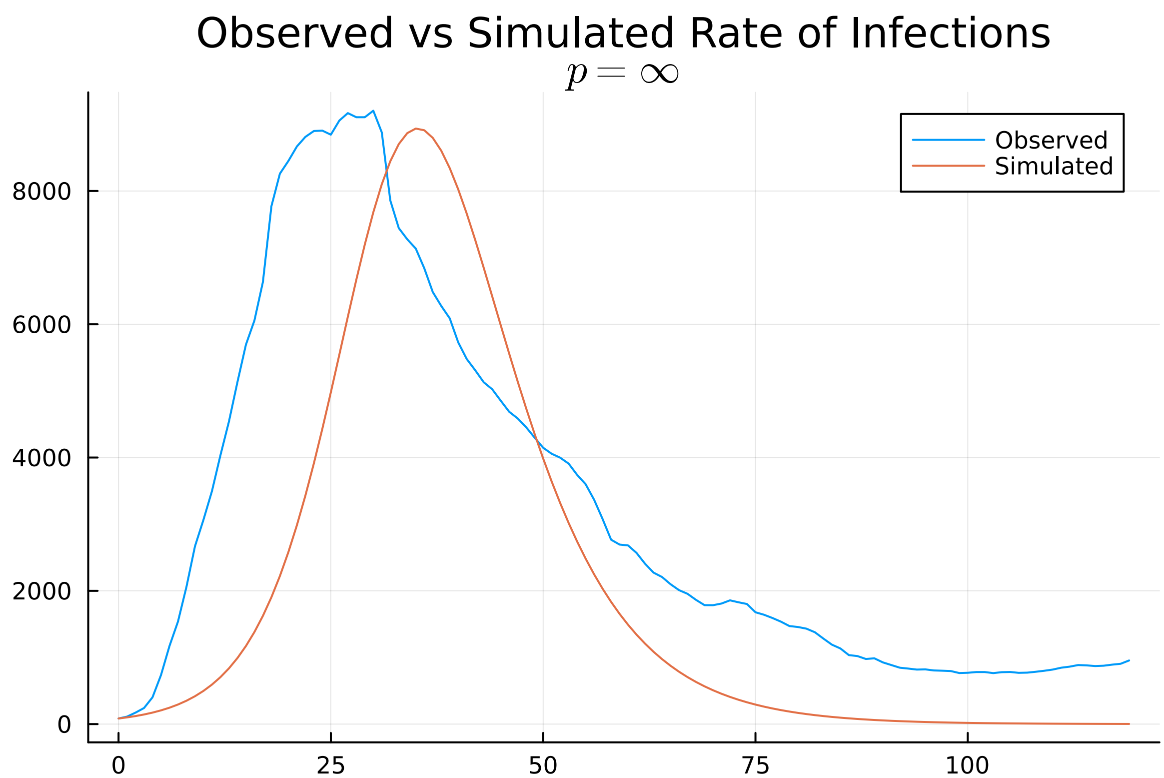


3)

In [37]:
```
Ss, Is, Rs = euler(α̂, β̂, N̂);
```

In [38]:
```
plot(0:119, I, labels="Observed")
plot!(0:119, Is, label="Simulated")
plot!(title = "Observed vs Simulated Rate of Infections" * "\n" * L"p=∞")
```

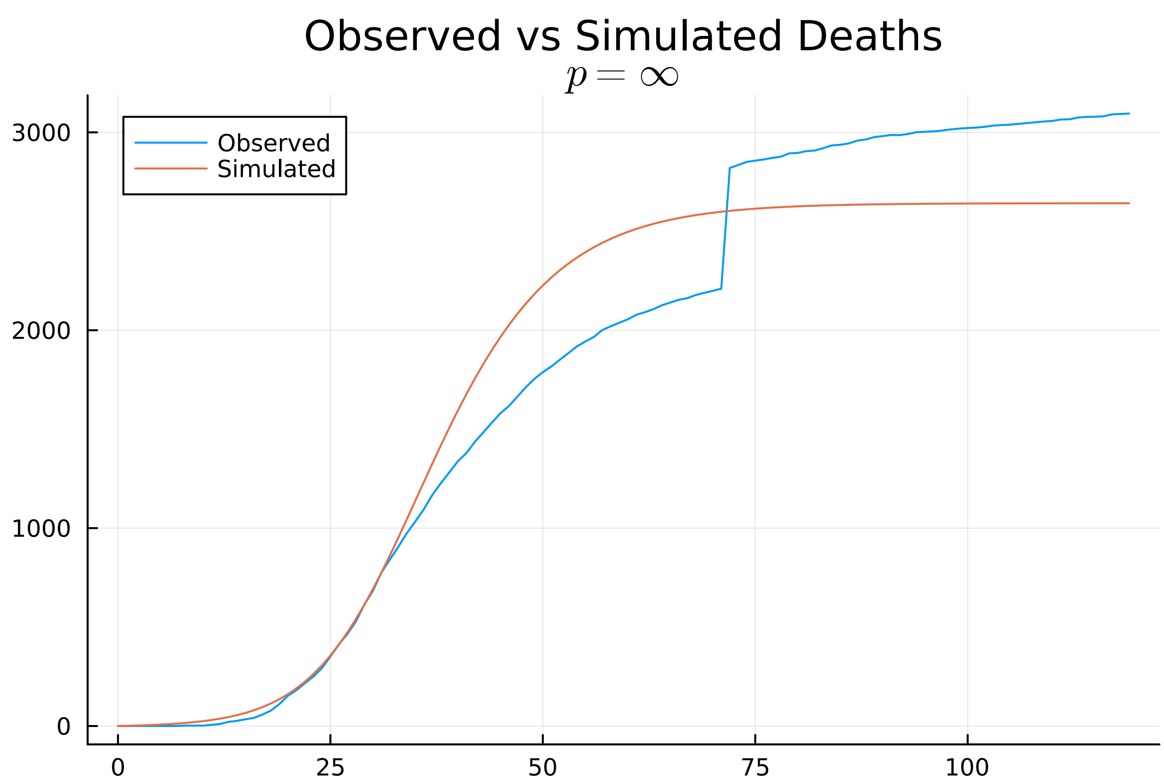
Out[38]:



In [39]:

```
plot(0:119, Y1, labels="Observed")
plot!(0:119, γ̂*Rs, label="Simulated")
plot!(title = "Observed vs Simulated Deaths" * "\n" * L"p=∞")
```

Out[39]:



4)

We would increase the value of α to maybe $\alpha \in [0.2, 0.6]$ to start. Also, an increase in $\hat{N}$ seems to return simulated values closer to the observed ones, so we would start by establishing $\frac{N}{Population} \in [10\%, 20\%]$. For R_0 we would increase its value since for all values of p we got $R_0 = \sup[1.2, 1.9]$, a good place to start would be $R_0 \in [1.9, 3]$

Excercise 2

In [40]:
```
# euler method to compute S_sim, E_sim, I_sim, and R_sim
function euler(alpha, beta, delta, N)
    h = 0.01

    S_sim = zeros(tmax+1)
    E_sim = zeros(tmax+1)
    I_sim = zeros(tmax+1)
    R_sim = zeros(tmax+1)

    s = N
    e = I(0)
    i = I(0)
    r = 0

    t = 0
    while t < tmax + 0.0001
      if abs(round(Int, t) - t) < 0.0001
        S_sim[round(Int, t)+1] = s
        E_sim[round(Int, t)+1] = e
        I_sim[round(Int, t)+1] = i
        R_sim[round(Int, t)+1] = r
      end

      ds = -beta*s*(i/N)
      de = beta*s*(i/N) - delta*e
      di = delta*e - alpha*i
      dr = alpha*i

      s += h*ds
      e += h*de
      i += h*di
      r += h*dr
      t += h
    end

    return S_sim, E_sim, I_sim, R_sim
  end
```

Out[40]: euler (generic function with 2 methods)

In [41]:
```
# objective function J
function J(alpha, beta, delta, N, gamma, p, I_sim, R_sim)
    is_inf = false
    if p == 3
       is_inf = true
```

```julia
        p = 1
    end

    s1(t) = abs(I(t) - I_sim[t+1])^p
    s2(t) = abs(Y(t) - gamma*R_sim[t+1])^p

    if is_inf
        return maximum(s1.(0:tmax) + λ*s2.(0:tmax))
    else
        return sum(s1, 0:tmax) + λ*sum(s2, 0:tmax)
    end
end
```

Out[41]: J (generic function with 1 method)

In [42]:

```julia
min = [Inf, Inf, Inf]
minimizer = [(-1.0,-1.0,-1.0,-1.0,-1.0), (-1.0,-1.0,-1.0,-1.0,-1.0), (-1.0,-

mind = [Dict(), Dict(), Dict()]

# run the euler method for each combination of parameters and find the minim
for alpha in ProgressBar(0.05:0.05:0.4)
    for beta in (1.5:0.1:1.9)*alpha
        for delta in 0.05:0.05:0.4
            for N in (2:10)*(population/100)
                I_sim, R_sim = euler(alpha, beta, delta, N)[3:4]

                for p in 1:3
                    gamma = optGamma(R_sim, p)
                    mind[p][(alpha, beta, delta, N, gamma)] = J(alpha, beta,

                    # if objective_function < min[p]
                    #     min[p] = objective_function
                    #     minimizer[p] = (alpha, beta, delta, N, gamma)
                    # end
                end
            end
        end
    end
end
```

```
0.0%|                                                | 0/8 [00:00<00:00, -0
s/it]
12.5%|██████                                          | 1/8 [00:12<Inf:Inf, InfG
s/it]
25.0%|████████████                                    | 2/8 [00:35<03:28, 35
s/it]
37.5%|██████████████████                              | 3/8 [01:01<02:33, 31
s/it]
50.0%|████████████████████████                        | 4/8 [01:29<01:58, 30
s/it]
62.5%|██████████████████████████████                  | 5/8 [01:57<01:28, 29
s/it]
75.0%|████████████████████████████████████            | 6/8 [02:25<00:58, 29
s/it]
87.5%|██████████████████████████████████████████      | 7/8 [02:54<00:29, 29
s/it]
100.0%|██████████████████████████████████████████████| 8/8 [03:23<00:00, 29
s/it]
100.0%|██████████████████████████████████████████████| 8/8 [03:23<00:00, 29
s/it]
```

In [43]:
```
for p in 1:3
    minimizer[p] = argmin(mind[p])
    min[p] = mind[p][minimizer[p]]
end
```

(1)

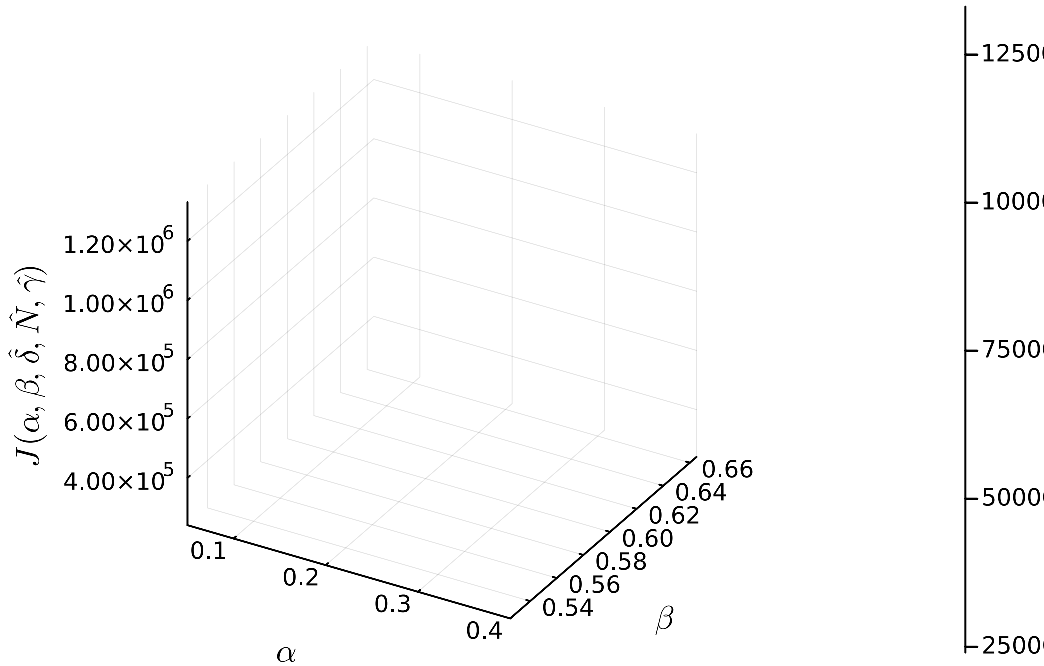
In [44]:
```
display_p = ["1", "2", "∞"]

# print out the min and minimizer for each value of p
for p in 1:3
    println("p = " * display_p[p] * ":")
    println("minimum of J(alpha, beta, delta, N, gamma) = $(min[p])")
    println("alpha = $(minimizer[p][1]), beta = $(minimizer[p][2]), delta =
    println("")
end
```

```
p = 1:
minimum of J(alpha, beta, delta, N, gamma) = 240107.84793462453
alpha = 0.35, beta = 0.6649999999999999, delta = 0.4, N = 81435.3, gamma =
0.04696363166542318

p = 2:
minimum of J(alpha, beta, delta, N, gamma) = 8.278247653300321e8
alpha = 0.4, beta = 0.76, delta = 0.4, N = 97722.36, gamma = 0.036569733756
936326

p = ∞:
minimum of J(alpha, beta, delta, N, gamma) = 7135.430012387385
alpha = 0.4, beta = 0.76, delta = 0.4, N = 162870.6, gamma = 0.021501287355
507076
```

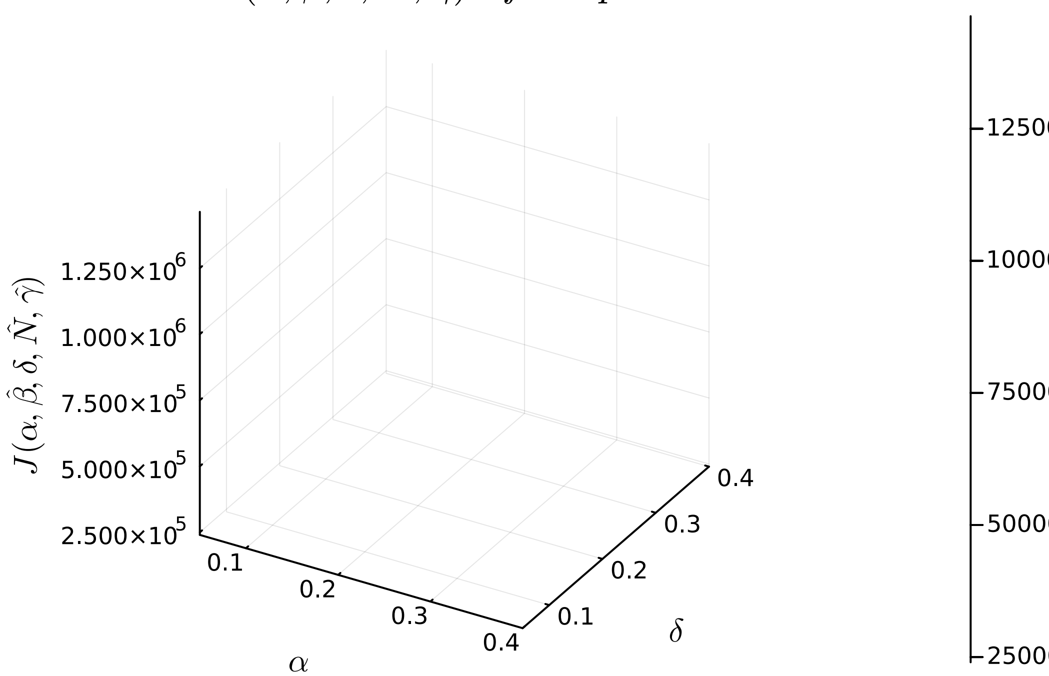
(2)

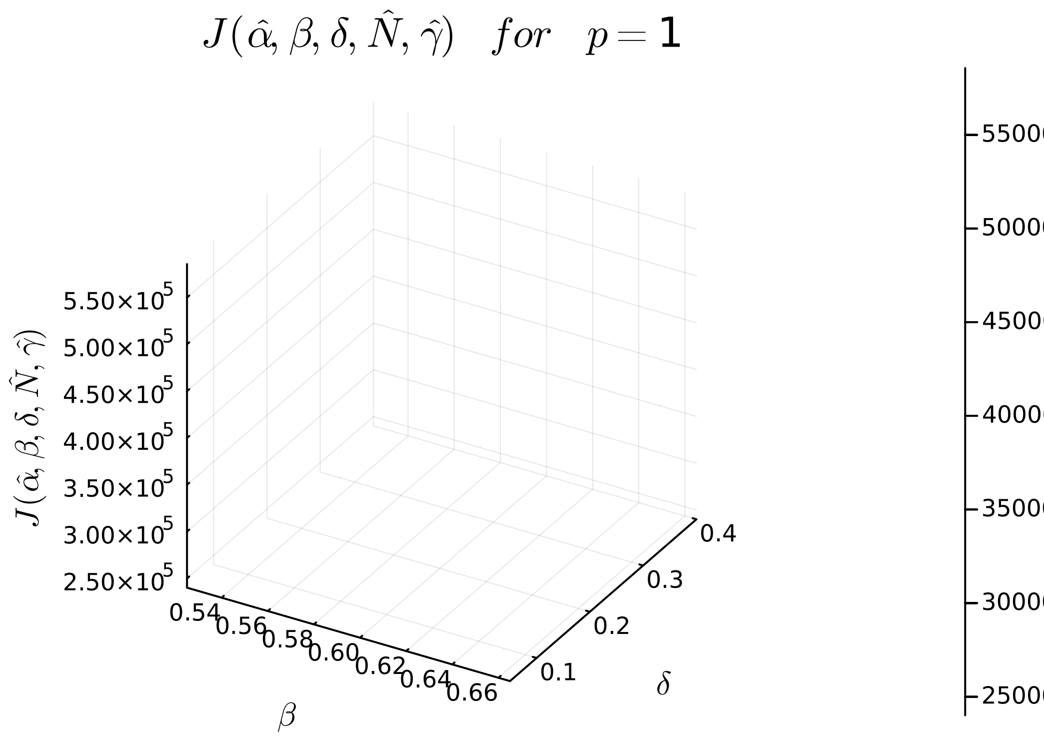
In [80]:
```
# make the 3 graphs of cross sections of J for each p
nnn = 1
for p in 1:3
    alpha, beta, delta, N, gamma = minimizer[p]

    function f_plot1(alpha, beta)
        S_sim, E_sim, I_sim, R_sim = euler(alpha, beta, delta, N)
        return J(alpha, beta, delta, N, gamma, p, I_sim, R_sim)
    end
    display(surface(0.05:0.05:0.4, (1.5:0.1:1.9)*alpha, f_plot1, label = ["a
        zlabel=L"J(\alpha, \beta, \hat{\delta}, \hat{N}, \hat{\gamma})", xla
    png("exercise2_plot$(nnn)")
    nnn+=1

    function f_plot2(alpha, delta)
        S_sim, E_sim, I_sim, R_sim = euler(alpha, beta, delta, N)
        return J(alpha, beta, delta, N, gamma, p, I_sim, R_sim)
    end
    display(surface(0.05:0.05:0.4, 0.05:0.05:0.4, f_plot2, label = ["alpha"
        zlabel=L"J(\alpha, \hat{\beta}, \delta, \hat{N}, \hat{\gamma})", xla
    png("exercise2_plot$(nnn)")
    nnn+=1

    function f_plot3(beta, delta)
        S_sim, E_sim, I_sim, R_sim = euler(alpha, beta, delta, N)
        return J(alpha, beta, delta, N, gamma, p, I_sim, R_sim)
    end
    display(surface((1.5:0.1:1.9)*alpha, 0.05:0.05:0.4, f_plot3, label = ["a
        zlabel=L"J(\hat{\alpha}, \beta, \delta, \hat{N}, \hat{\gamma})", xla
    png("exercise2_plot$(nnn)")
    nnn+=1
end
```

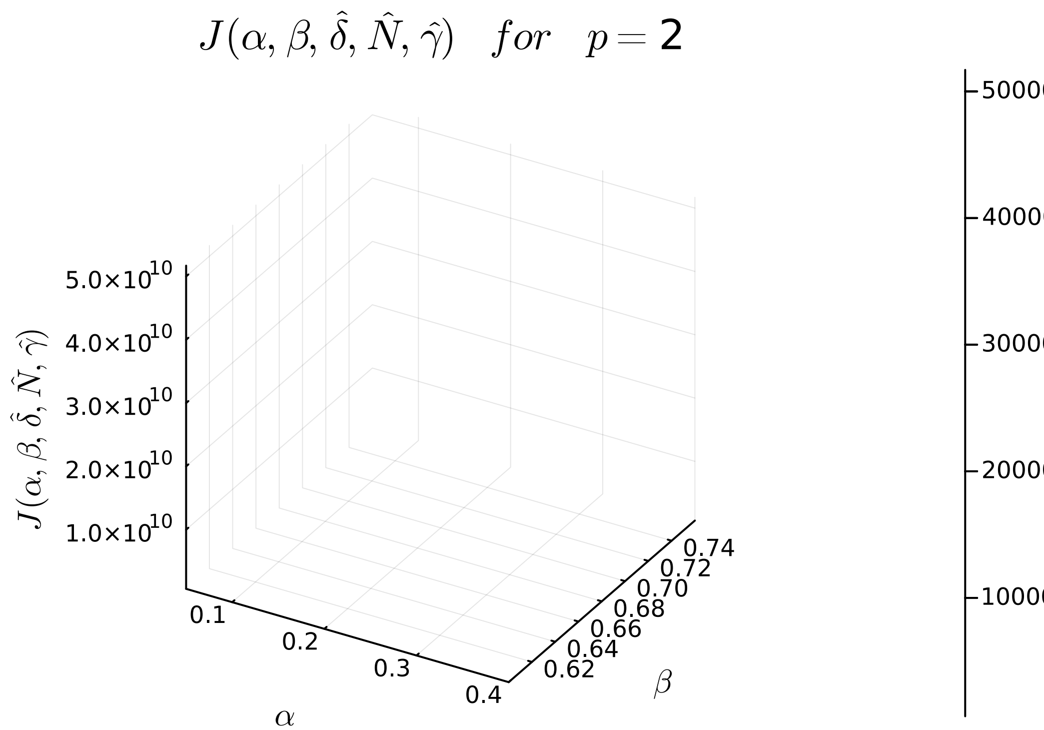
$J(\alpha, \beta, \hat{\hat{\delta}}, \hat{N}, \hat{\gamma}) \quad for \quad p = 1$

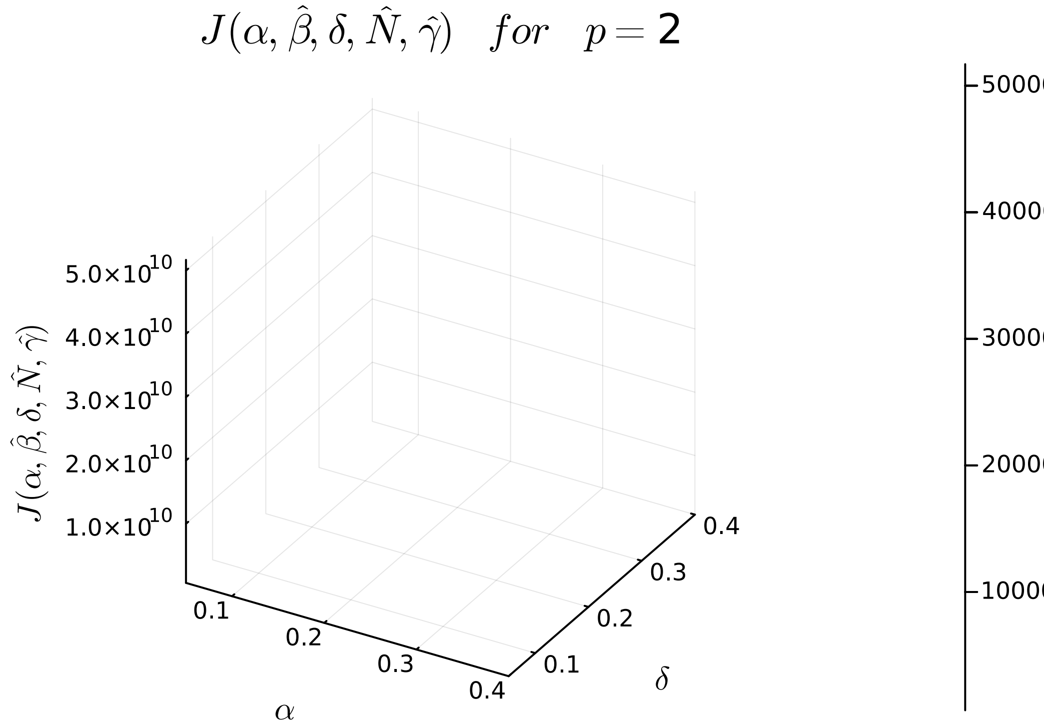


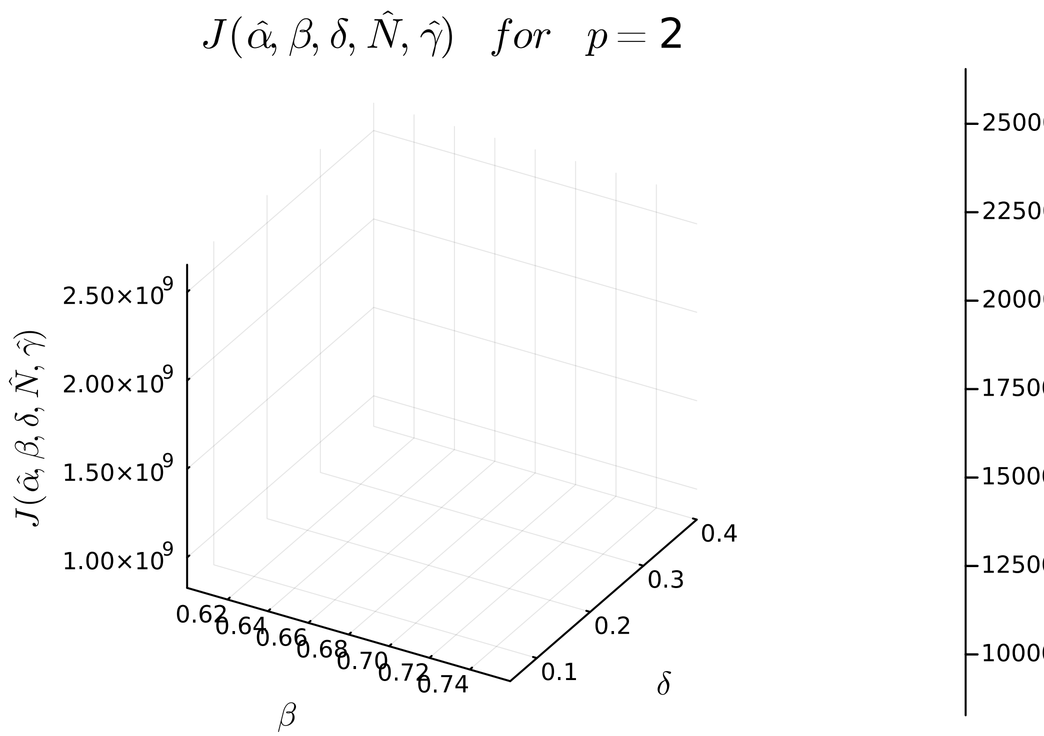
$J(\alpha, \hat{\beta}, \delta, \hat{N}, \hat{\gamma}) \quad for \quad p = 1$



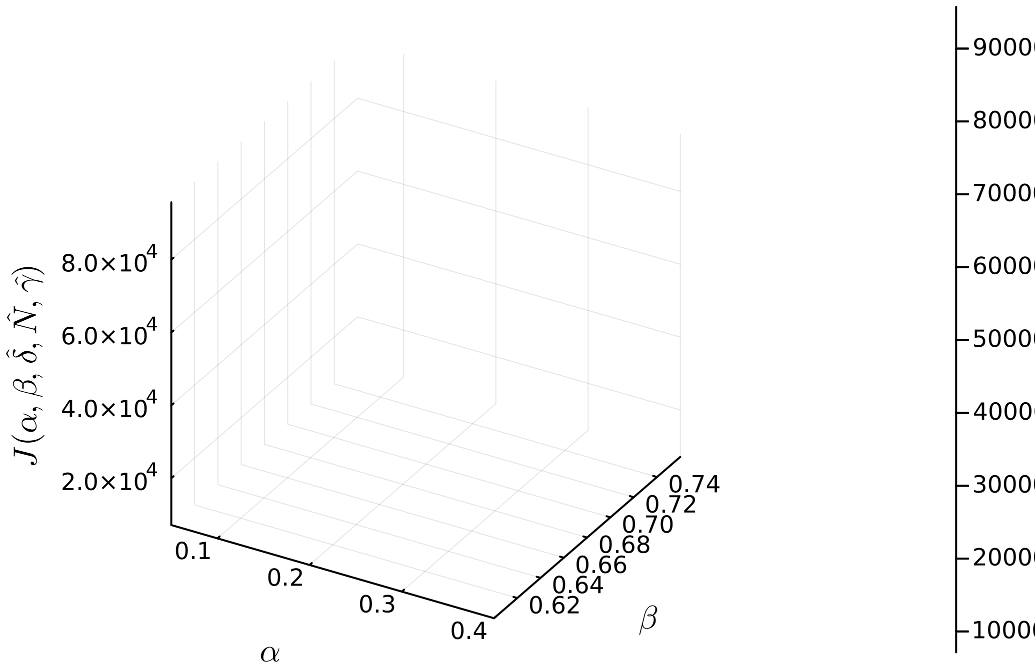

$J(\hat{\alpha}, \beta, \delta, \hat{N}, \hat{\gamma}) \quad for \quad p = 1$
$J(\hat{\alpha}, \beta, \delta, \hat{N}, \hat{\gamma})$
5.50×10^5
5.00×10^5
4.50×10^5
4.00×10^5
3.50×10^5
3.00×10^5
2.50×10^5
0.54 0.56 0.58 0.60 0.62 0.64 0.66
β
0.1 0.2 0.3 0.4
δ
55000
50000
45000
40000
35000
30000
25000


$J(\alpha, \beta, \hat{\delta}, \hat{N}, \hat{\gamma}) \quad for \quad p = 2$
$J(\alpha, \beta, \hat{\delta}, \hat{N}, \hat{\gamma})$
5.0×10^10
4.0×10^10
3.0×10^10
2.0×10^10
1.0×10^10
0.1 0.2 0.3 0.4
α
0.62 0.64 0.66 0.68 0.70 0.72 0.74
β
50000
40000
30000
20000
10000

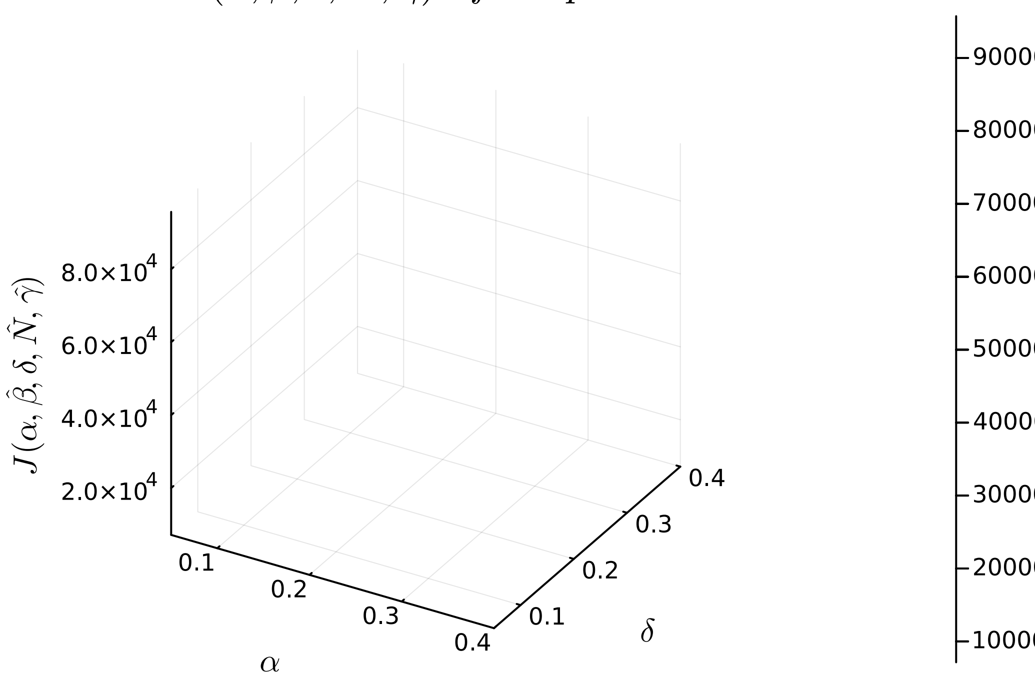

$J(\alpha, \hat{\beta}, \delta, \hat{N}, \hat{\gamma}) \quad for \quad p = 2$
$J(\alpha, \hat{\beta}, \delta, \hat{N}, \hat{\gamma})$
5.0×10^{10}
4.0×10^{10}
3.0×10^{10}
2.0×10^{10}
1.0×10^{10}
0.1
0.2
0.3
0.4
α
0.1
0.2
0.3
0.4
δ
50000
40000
30000
20000
10000

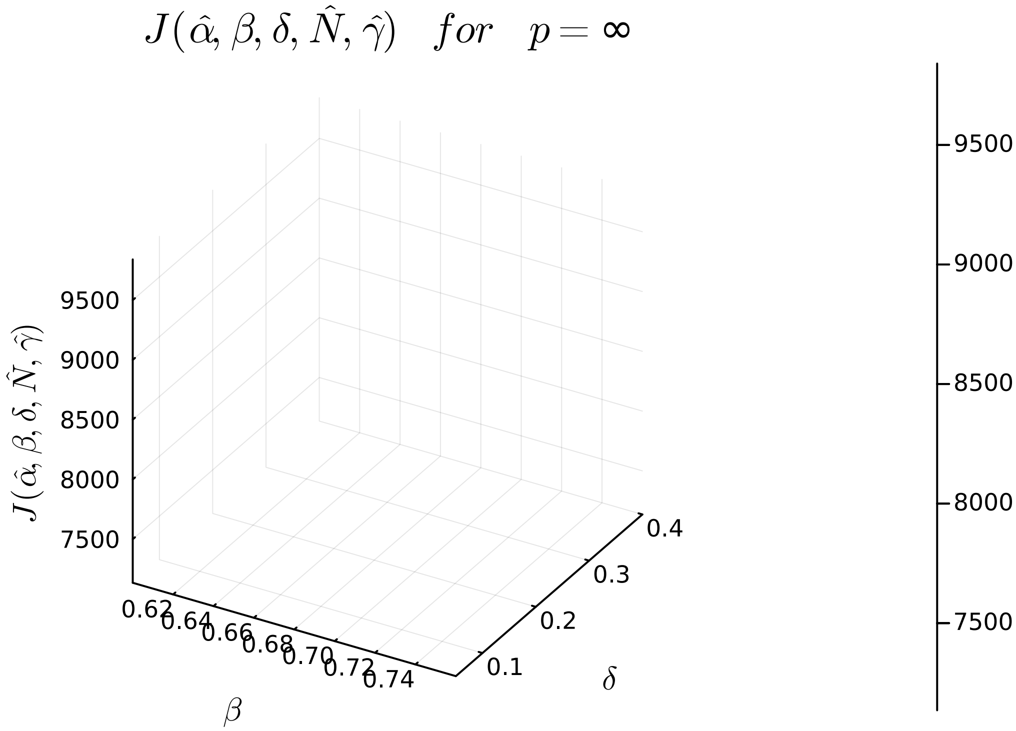

$J(\hat{\alpha}, \beta, \delta, \hat{N}, \hat{\gamma}) \quad for \quad p = 2$
$J(\hat{\alpha}, \beta, \delta, \hat{N}, \hat{\gamma})$
2.50×10^{9}
2.00×10^{9}
1.50×10^{9}
1.00×10^{9}
0.62
0.64
0.66
0.68
0.70
0.72
0.74
β
0.1
0.2
0.3
0.4
δ
25000
22500
20000
17500
15000
12500
10000

$$J(\alpha, \beta, \hat{\delta}, \hat{N}, \hat{\gamma}) \quad for \quad p = \infty$$



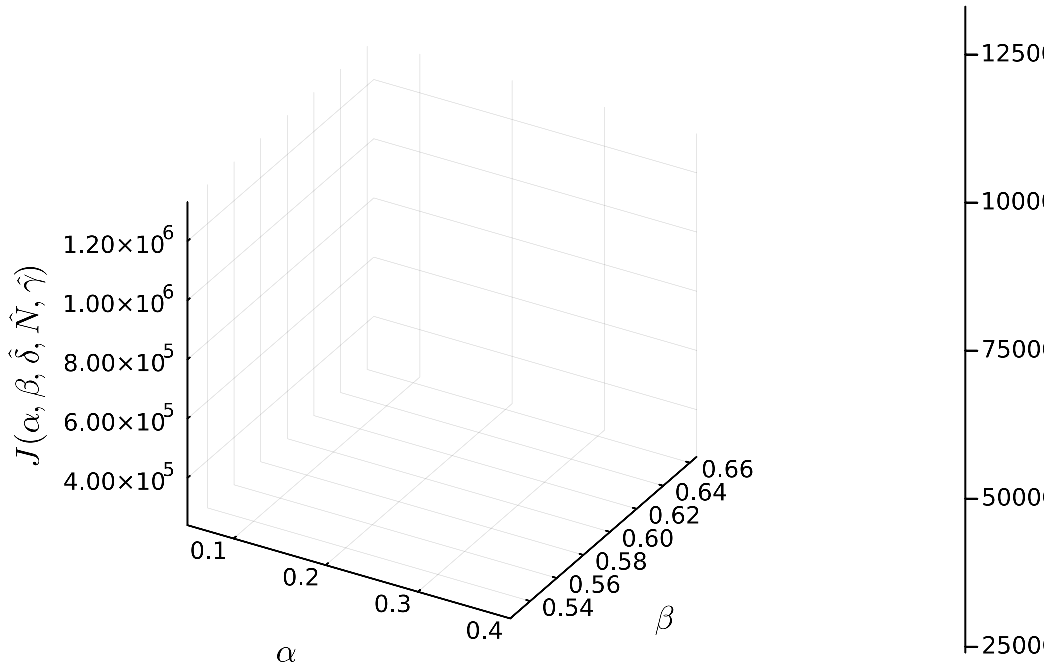
$$J(\alpha, \hat{\beta}, \delta, \hat{N}, \hat{\gamma}) \quad for \quad p = \infty$$



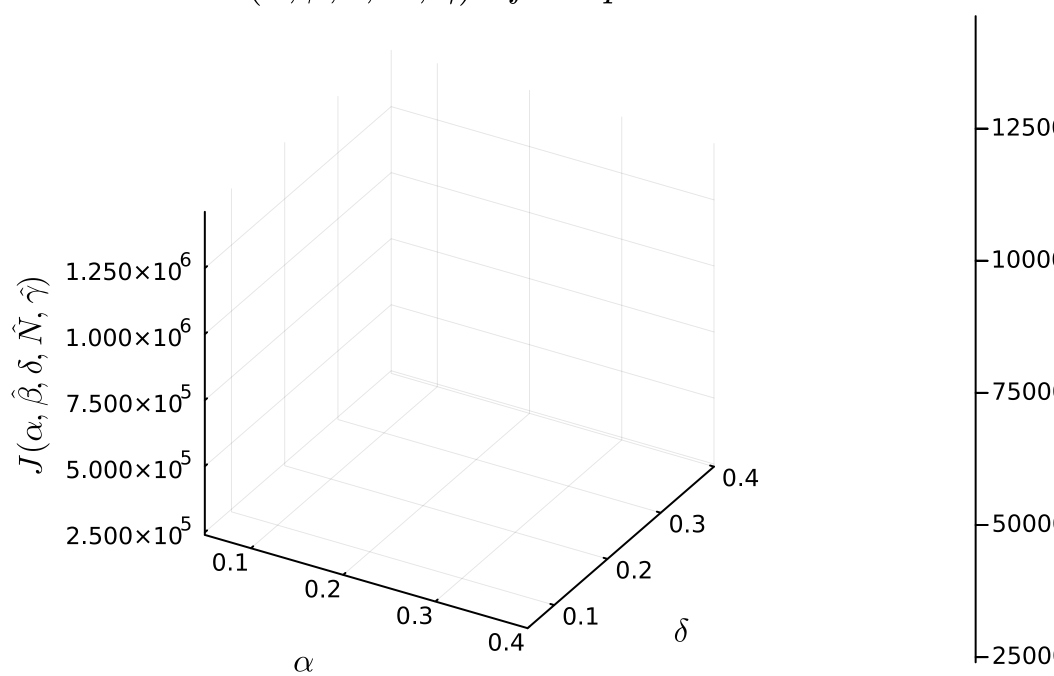


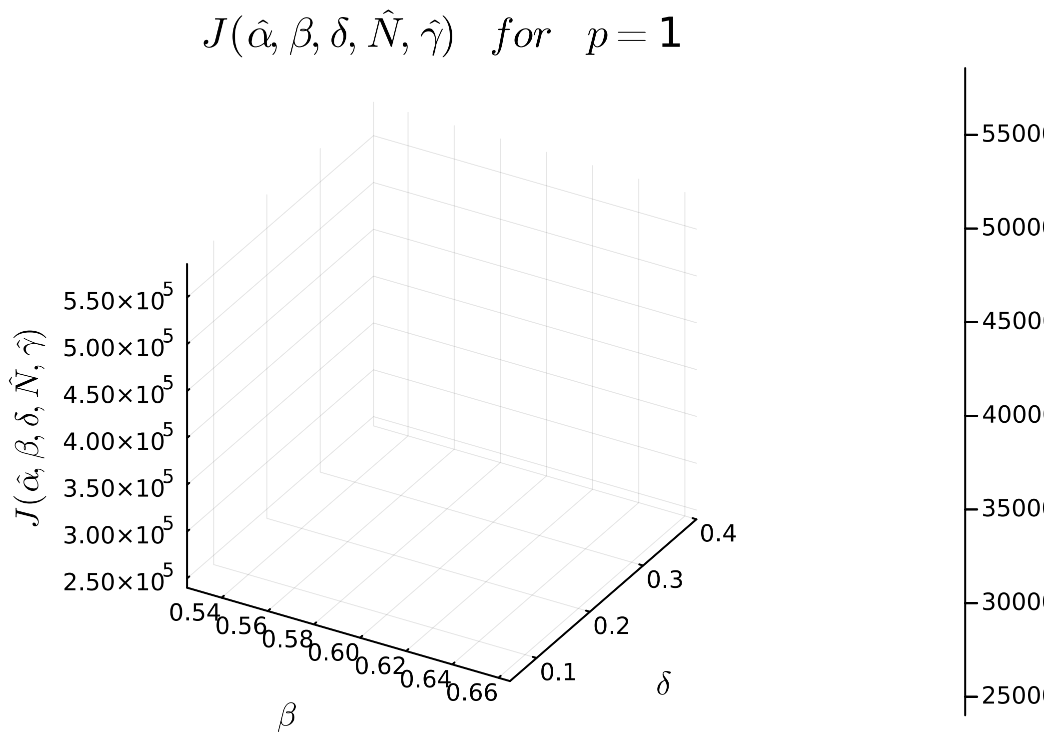

```
In [74]: # make the 3 graphs of cross sections of J for each p
for p in 1:3
    alpha, beta, delta, N, gamma = minimizer[p]

    function f_plot1(alpha, beta)
        S_sim, E_sim, I_sim, R_sim = euler(alpha, beta, delta, N)
        return J(alpha, beta, delta, N, gamma, p, I_sim, R_sim)
    end
    display(surface(0.05:0.05:0.4, (1.5:0.1:1.9)*alpha, f_plot1, label = ["a
        zlabel=L"J(\alpha, \beta, \hat{\delta}, \hat{N}, \hat{\gamma})", xla

    function f_plot2(alpha, delta)
        S_sim, E_sim, I_sim, R_sim = euler(alpha, beta, delta, N)
        return J(alpha, beta, delta, N, gamma, p, I_sim, R_sim)
    end
    display(surface(0.05:0.05:0.4, 0.05:0.05:0.4, f_plot2, label = ["alpha"
        zlabel=L"J(\alpha, \hat{\beta}, \delta, \hat{N}, \hat{\gamma})", xla

    function f_plot3(beta, delta)
        S_sim, E_sim, I_sim, R_sim = euler(alpha, beta, delta, N)
        return J(alpha, beta, delta, N, gamma, p, I_sim, R_sim)
    end
    display(surface((1.5:0.1:1.9)*alpha, 0.05:0.05:0.4, f_plot3, label = ["a
        zlabel=L"J(\hat{\alpha}, \beta, \delta, \hat{N}, \hat{\gamma})", xla
end
```

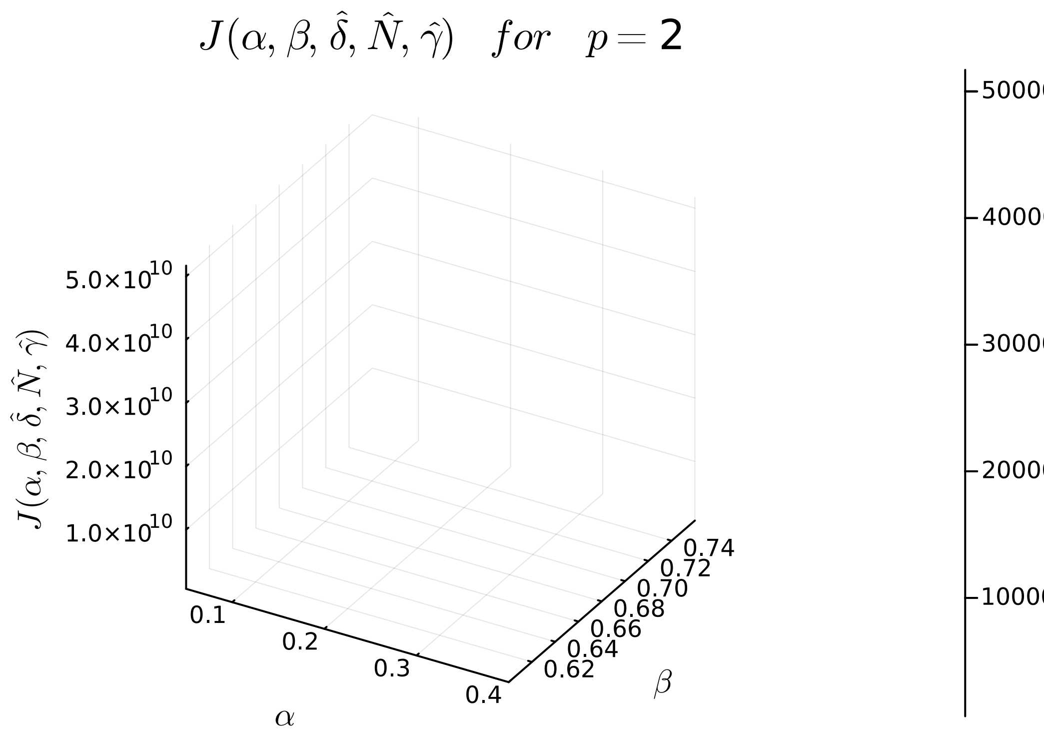
$J(\alpha, \beta, \hat{\hat{\delta}}, \hat{N}, \hat{\gamma}) \quad for \quad p = 1$

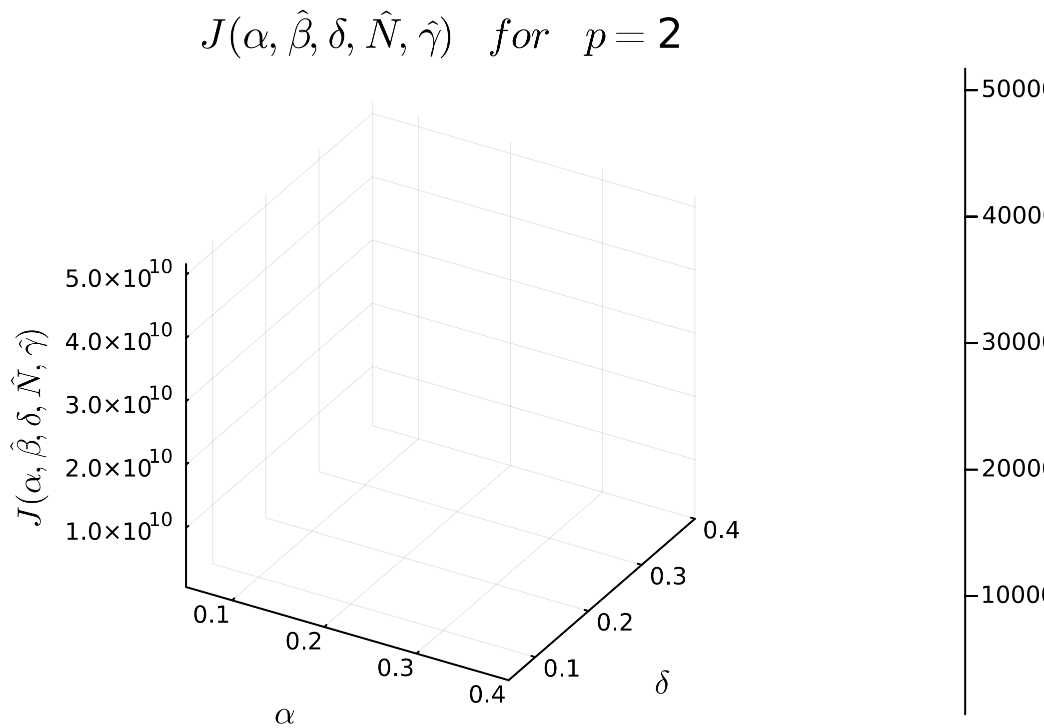


$J(\alpha, \hat{\hat{\beta}}, \delta, \hat{N}, \hat{\gamma}) \quad for \quad p = 1$

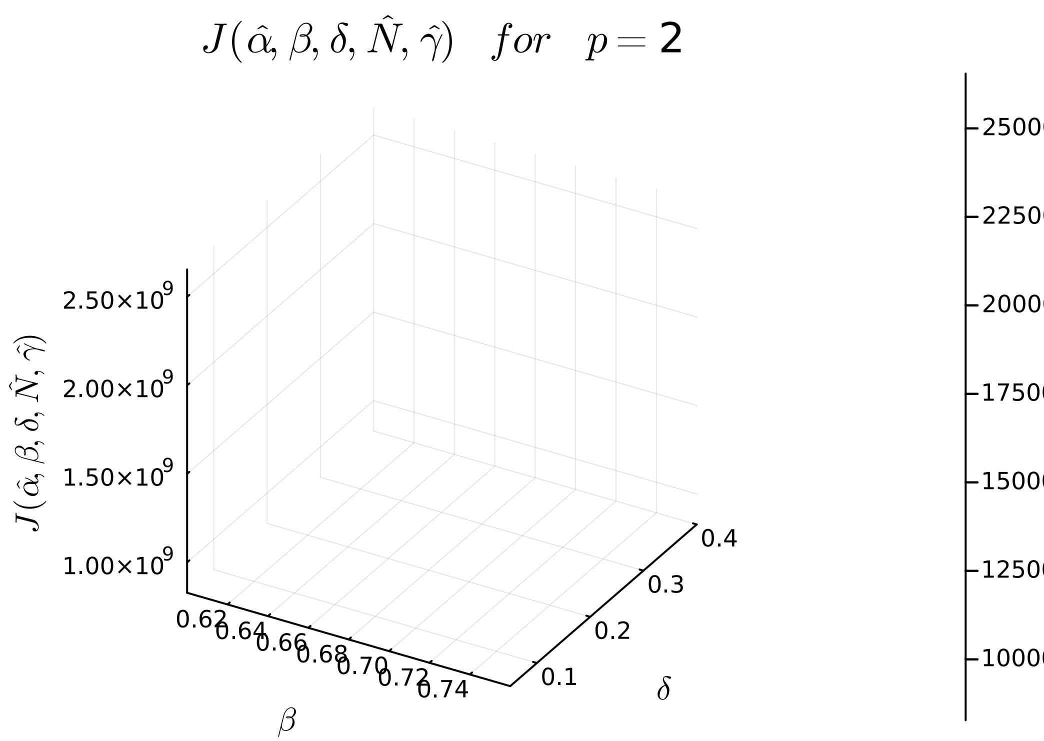



$J(\hat{\alpha}, \beta, \delta, \hat{N}, \hat{\gamma}) \quad for \quad p = 1$
5.50×10^5
5.00×10^5
4.50×10^5
4.00×10^5
3.50×10^5
3.00×10^5
2.50×10^5
$J(\hat{\alpha}, \beta, \delta, \hat{N}, \hat{\gamma})$
0.54
0.56
0.58
0.60
0.62
0.64
0.66
β
0.1
0.2
0.3
0.4
δ
55000
50000
45000
40000
35000
30000
25000

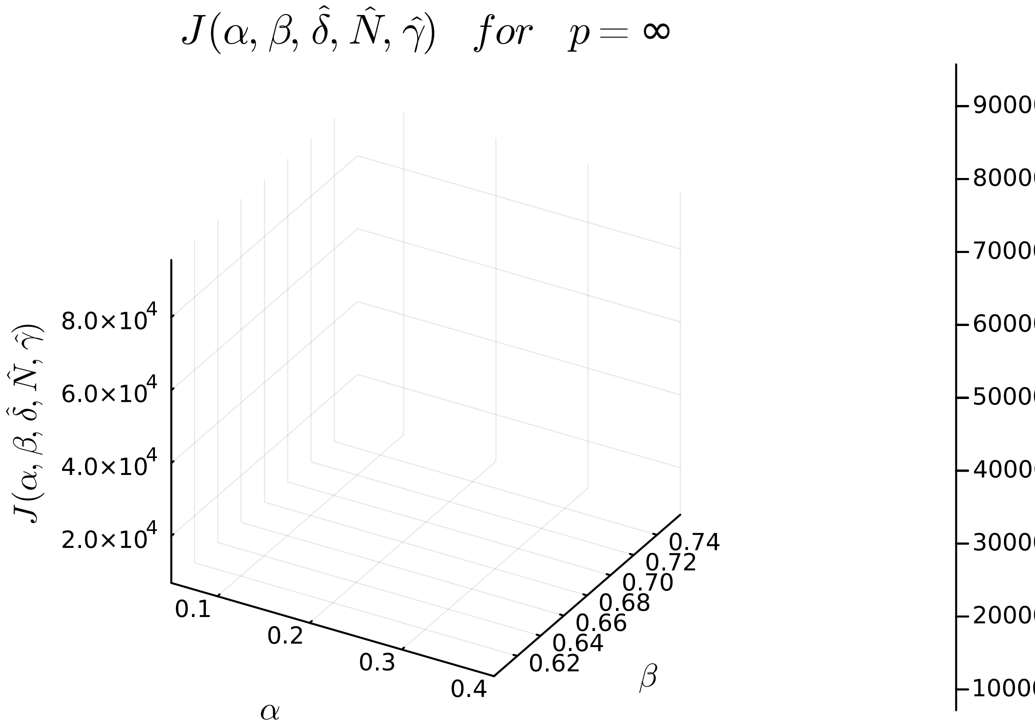

$J(\alpha, \beta, \hat{\delta}, \hat{N}, \hat{\gamma}) \quad for \quad p = 2$
5.0×10^10
4.0×10^10
3.0×10^10
2.0×10^10
1.0×10^10
$J(\alpha, \beta, \hat{\delta}, \hat{N}, \hat{\gamma})$
0.1
0.2
0.3
0.4
α
0.62
0.64
0.66
0.68
0.70
0.72
0.74
β
50000
40000
30000
20000
10000

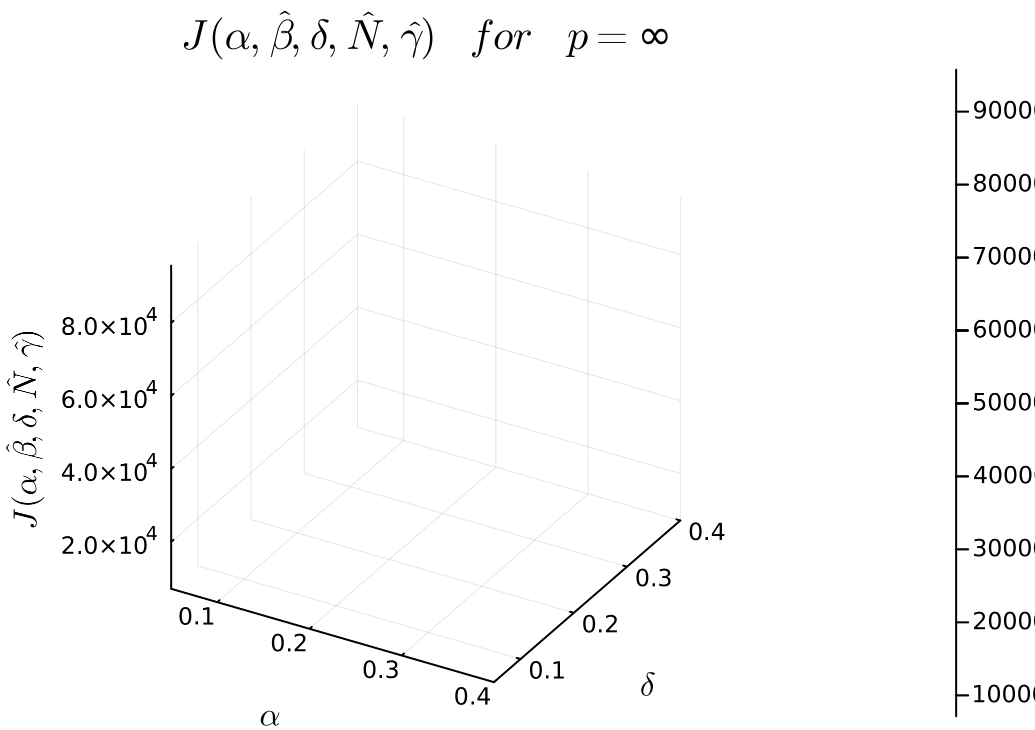


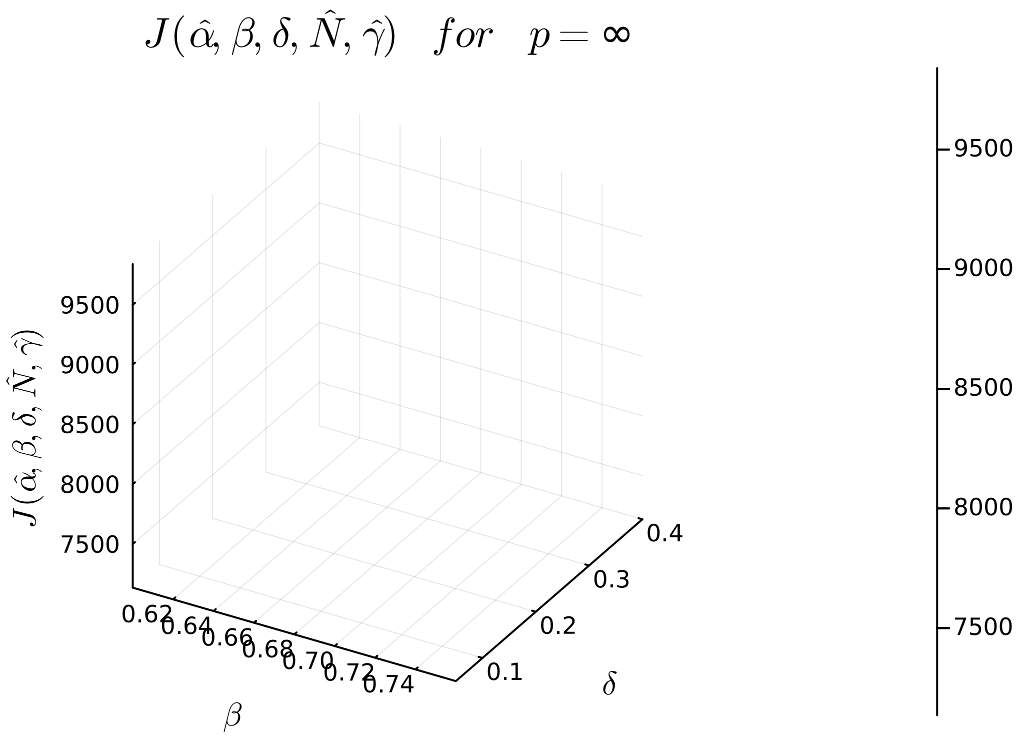
$J(\alpha, \hat{\beta}, \delta, \hat{N}, \hat{\gamma}) \quad for \quad p = 2$
$J(\alpha, \hat{\beta}, \delta, \hat{N}, \hat{\gamma})$
α
δ



$J(\hat{\alpha}, \beta, \delta, \hat{N}, \hat{\gamma}) \quad for \quad p = 2$
$J(\hat{\alpha}, \beta, \delta, \hat{N}, \hat{\gamma})$
β
δ


$J(\alpha, \beta, \hat{\delta}, \hat{N}, \hat{\gamma}) \quad for \quad p = \infty$
$J(\alpha, \beta, \hat{\delta}, \hat{N}, \hat{\gamma})$
8.0×10^4
6.0×10^4
4.0×10^4
2.0×10^4
0.1
0.2
0.3
0.4
α
0.62
0.64
0.66
0.68
0.70
0.72
0.74
β
90000
80000
70000
60000
50000
40000
30000
20000
10000

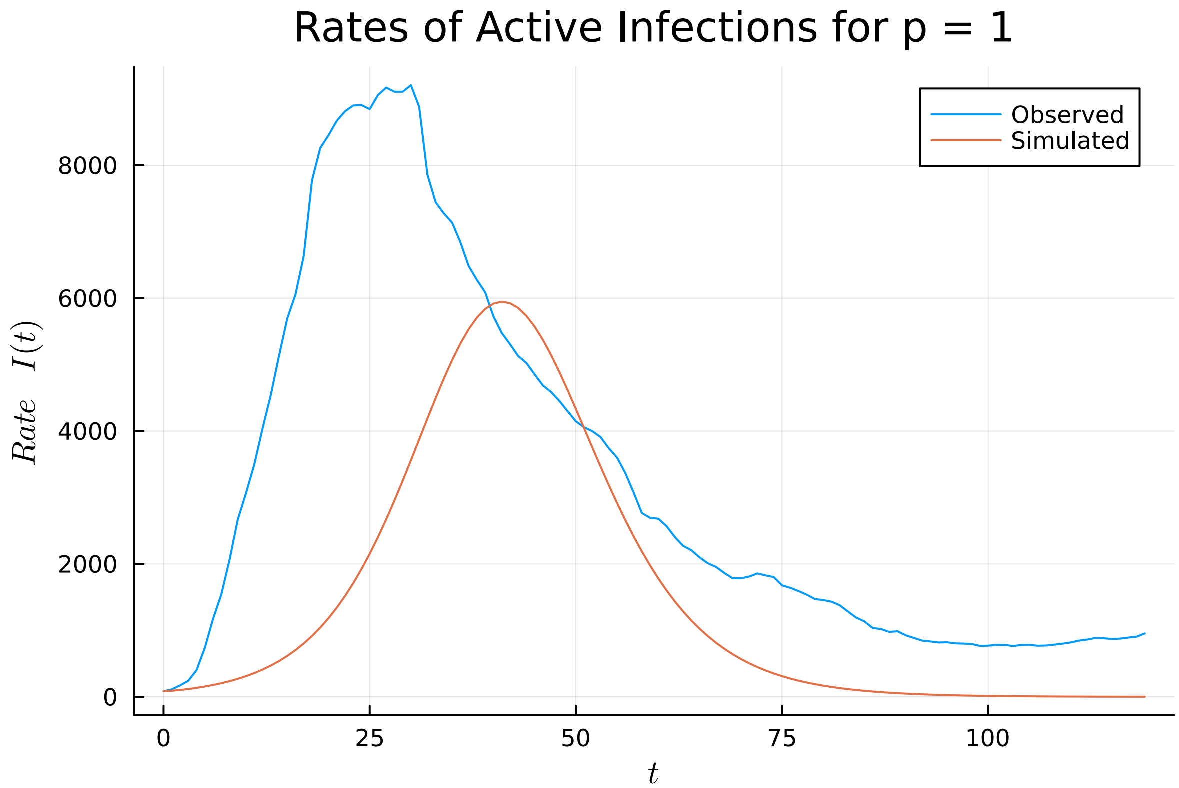

$J(\alpha, \hat{\beta}, \delta, \hat{N}, \hat{\gamma}) \quad for \quad p = \infty$
$J(\alpha, \hat{\beta}, \delta, \hat{N}, \hat{\gamma})$
8.0×10^4
6.0×10^4
4.0×10^4
2.0×10^4
0.1
0.2
0.3
0.4
α
0.1
0.2
0.3
0.4
δ
90000
80000
70000
60000
50000
40000
30000
20000
10000

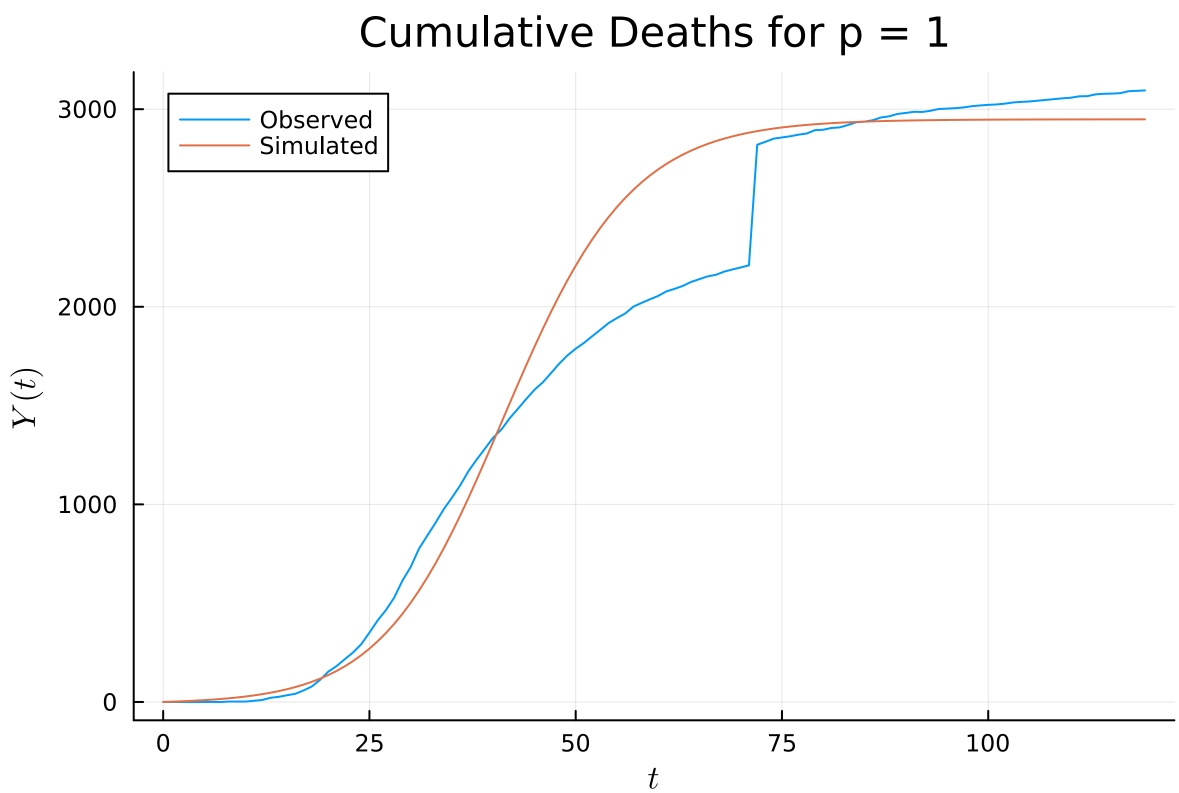


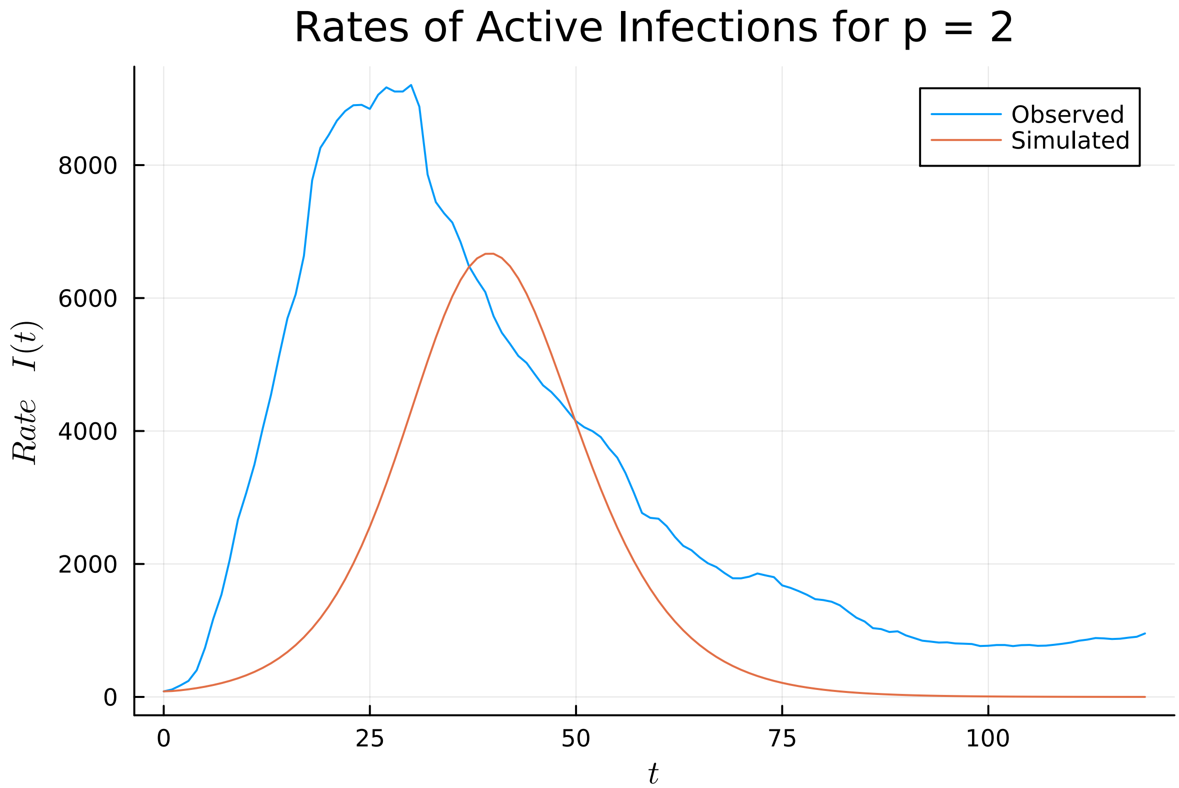
(3)

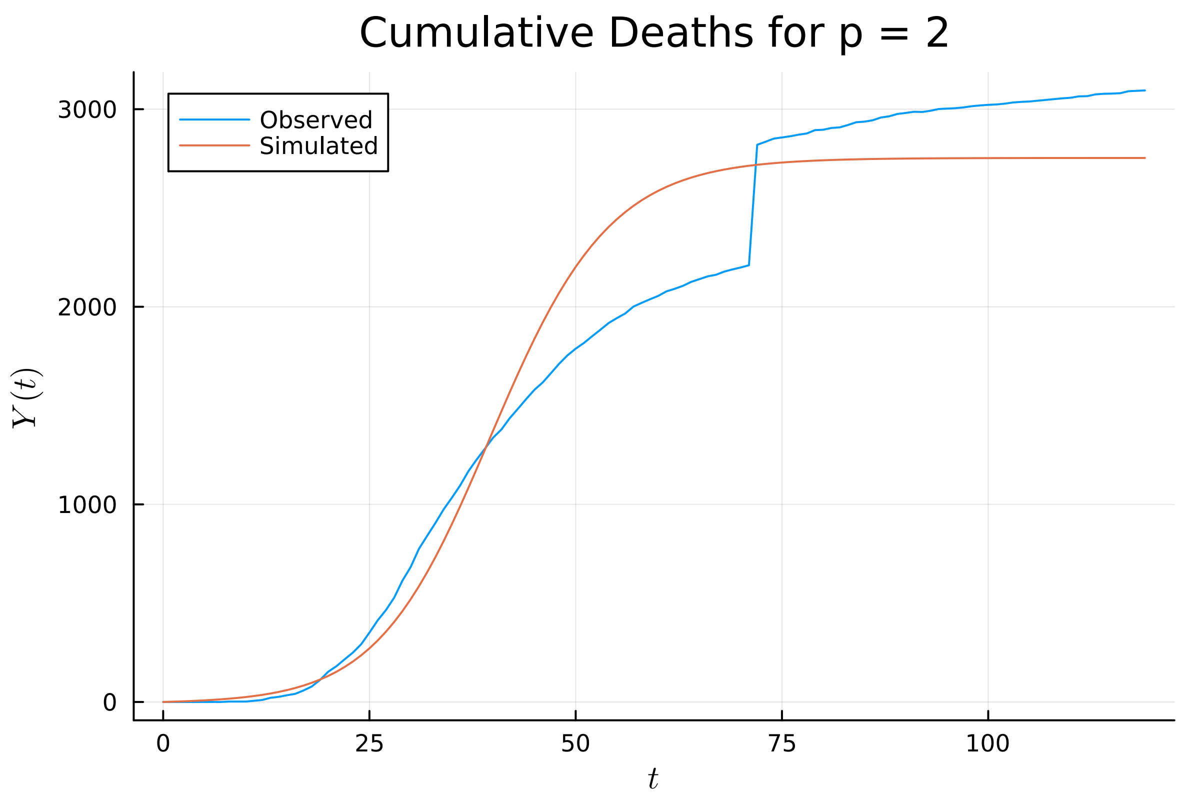
In [46]:

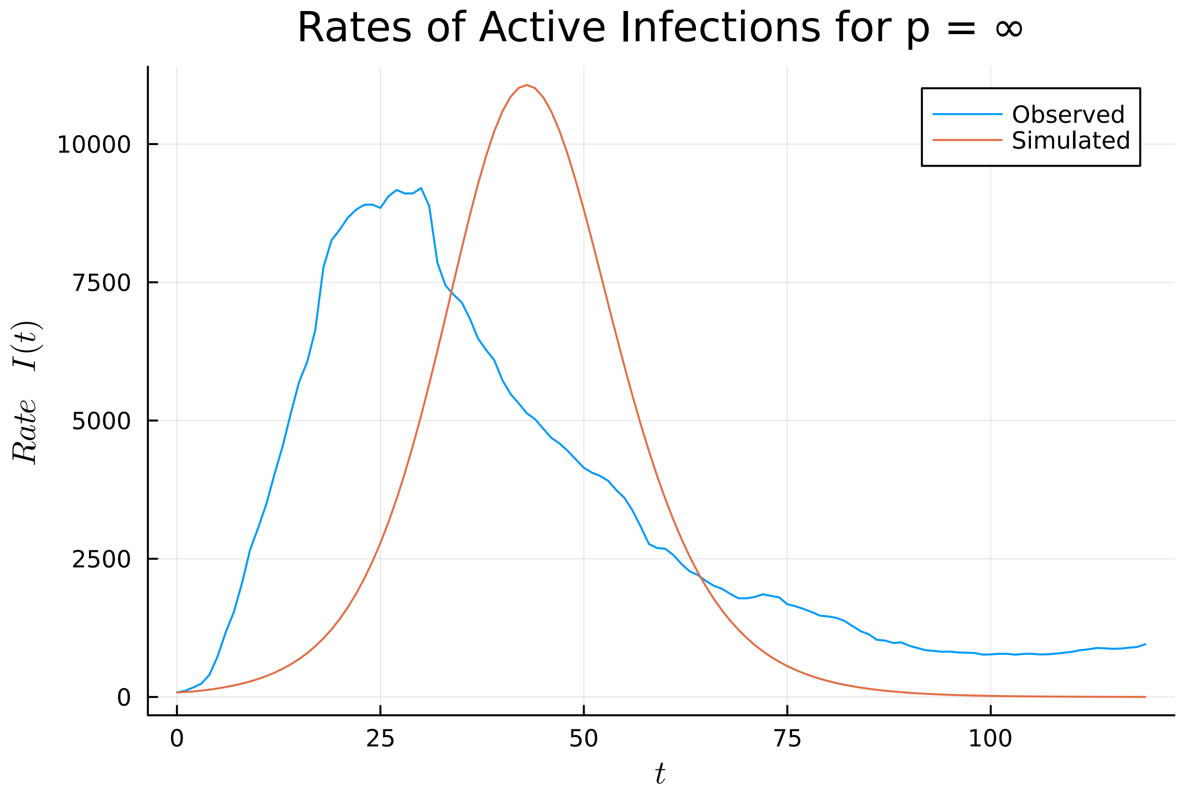
```
# plot I vs I_sim and Y vs gamma*R_sim
for p = 1:3
    alpha, beta, delta, N, gamma = minimizer[p]
    S_sim, E_sim, I_sim, R_sim = euler(alpha, beta, delta, N)

    display(plot(0:tmax, [I.(0:tmax), I_sim], label = ["Observed" "Simulated
    display(plot(0:tmax, [Y.(0:tmax), gamma*R_sim], label = ["Observed" "Sim
end
```

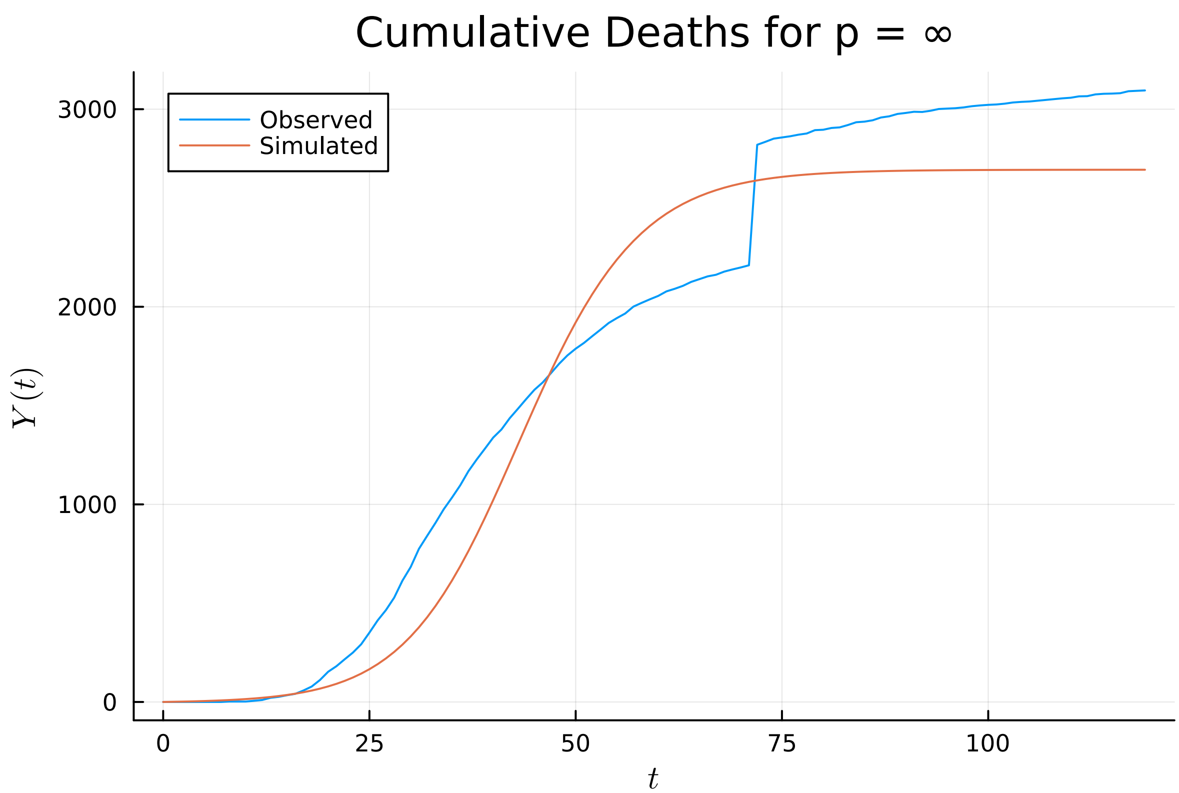

Rates of Active Infections for p = 1
Observed
Simulated
8000
6000
4000
2000
0
0
25
50
75
100
$Rate \quad I(t)$
t


Cumulative Deaths for p = 1
Observed
Simulated
3000
2000
1000
0
0
25
50
75
100
$Y(t)$
t


Rates of Active Infections for p = 2
Observed
Simulated
$Rate \quad I(t)$
8000
6000
4000
2000
0
0
25
50
75
100
t


Cumulative Deaths for p = 2
Observed
Simulated
$Y(t)$
3000
2000
1000
0
0
25
50
75
100
t





(4)

In all of the objective fucntions with alpha the low values of alpha make the function very large and hard to tell what is happening in the low part. So narrowing in on the optimal alpha and seeing the effect delta and beta have in this reduced range would be good.

For the graphs with beta and delta all three have there lowest point in the corner so it would be a good idea to increase the top of the range being searched for beta and delta. considering this some ranges to check would be alpha in [0.3, 0.4], R0 in [1.5, 3], delta in [0.05, 1], N/population in [2%, 10%].