Quantitative Economics_HW5

April 5, 2022

In the name of God

[72]:
```
using  Plots, LinearAlgebra, GLM, Optim, DataFrames, StatFiles, Distributions,␣
 ↪Roots, KernelDensity, BlackBoxOptim
pyplot();
```

1 Monte-Carlo Simulation of firms behavior

1.1

First of all, to simplify the problem we solve the cost minimization problem for the firm. (our goal is to get rid of L :)) To reach our aim we have two ways: 1) from the question's assumptions we have:

$$y = Al^{\alpha}$$

so we can easily simplify the equation and then we have:

$$l^* = (\frac{y}{A})^{\frac{1}{\alpha}}$$

2) we can classically solve the CMP:

$$\min wl + f_0 s.t. y <= Al^{\alpha} \Rightarrow Lagrangian = -wl - f_0 + \lambda(Al^{\alpha} - y) \Rightarrow [l] : -w + \lambda(\alpha A l^{\alpha-1}) \Rightarrow l^* = (\frac{w}{\lambda\alpha A})^{\frac{1}{\alpha-1}}$$

with some algebra …:

$$\Rightarrow l^* = (\frac{y}{A})^{\frac{1}{\alpha}}$$

So, to set up the firm's problem we want to set up the profit function:

$$\pi = total\ revenu - total\ cost \pi = py - (wl^* + f_0)\pi = py - (w(\frac{y}{A})^{\frac{1}{\alpha}} + f_0)$$

Now we can set up a profit maximization problem for the firm:

$$\max : py - (w(\frac{y}{A})^{\frac{1}{\alpha}} + f_0) s.t.\ y = Y(\frac{p}{P})^{-\sigma}$$

1.2

We can rewrite the problem as below:

$$\max: py - Cost(y) s.t.\ y = Y(\frac{p}{P})^{-\sigma}$$

Now we want to solve the problem for price. we can easily get derivative from objective function with respect to price.

$$\Rightarrow\ y + p\frac{dy}{dp} - \frac{dy}{dp}Cost^{'}(y) = 0$$

We remember from microeconomics that term $\frac{\frac{dy}{y}}{\frac{dp}{p}}$ is equal to price elasticity of demand. So with some algebraic calculation, we have:

$$\Rightarrow\ y\ (1 + \frac{\frac{dy}{y}}{\frac{dp}{p}} - \frac{\frac{dy}{y}}{\frac{dp}{p}}\frac{Cost^{'}(y)}{p}) = 0$$

From the functional form $y = Y(\frac{p}{P})^{-\sigma}$ we know that the price elasticity of demand is equal to $-\sigma$. obvously we can calculate the term $\frac{\frac{dy}{y}}{\frac{dp}{p}}$ from the $y = Y(\frac{p}{P})^{-\sigma}$ and see the result.

So, we can rewrite the derivative as below:

$$y\ (1 - \sigma - Cost^{'}(y)\frac{-\sigma}{p}) = 0$$

with sum algebraic calculation we have:

$$p^{*}\ =\ \frac{\sigma}{\sigma - 1}Cost^{'}(y)$$

Also, we can rewrite the p^{*} with substitution of $Cost^{'}(y)$ as below:

$$Cost^{'}(y) = \frac{w}{a}(\frac{1}{A})^{\frac{1}{\alpha}}\ y^{\frac{1-\alpha}{\alpha}}$$

We know that $y = Y(\frac{p}{P})^{-\sigma}$ So:

$$p^{*} = \frac{\sigma}{\sigma - 1}\frac{w}{a}(\frac{1}{A})^{\frac{1}{\alpha}}\ (Y(\frac{p}{P})^{-\sigma})^{\frac{1-\alpha}{\alpha}} p^{*} = (\frac{\sigma}{\sigma - 1})(\frac{w}{a})(\frac{1}{A})^{\frac{1}{\alpha}}(Y^{\frac{1-\alpha}{\alpha}})(P^{\frac{\sigma-\alpha\sigma}{\alpha}})(p^{*\frac{\sigma\alpha-\sigma}{\alpha}})p^{*} = ((\frac{\sigma}{\sigma - 1})(\frac{w}{a})(\frac{1}{A})^{\frac{1}{\alpha}}(Y^{\frac{1-\alpha}{\alpha}})(P$$

1.3

We can calculate the threshold of A by solving the \$ = 0 \$ equation.

$$\Rightarrow \pi = py - (w(\frac{y}{A})^{\frac{1}{\alpha}} + f_0)\ =\ 0$$

$$\Rightarrow py = w(\frac{y}{A})^{\frac{1}{\alpha}} + f_0$$

$$\Rightarrow py - f_0 = w\frac{y^{\frac{1}{\alpha}}}{A^{\frac{1}{\alpha}}}$$

$$\Rightarrow \frac{py - f_0}{wy^{\frac{1}{\alpha}}} = \frac{1}{A^{\frac{1}{\alpha}}}$$

$$\Rightarrow (\frac{py - f_0}{wy^{\frac{1}{\alpha}}})^{\alpha} = \frac{1}{A}$$

$$\Rightarrow A = (\frac{wy^{\frac{1}{\alpha}}}{py - f_0})^{\alpha}$$

$$\Rightarrow A = \frac{w^{\alpha} y}{(py - f_0)^{\alpha}}$$

1.4

[73]:
```
function firm( , A_bar,  , P, Y,  , f0, w; N=1000)
    A = rand(Pareto( ,A_bar),N)
    p = ((  ./ ( -1)) .* (w ./  ) .* ((1 ./ A) .^ (1 ./  )) .* (Y .^ ((1- ) ./  ))␣
  ↪.* (P .^ ( *(1- ) ./  ))) .^ (  ./ ( - *( -1)))
    y = Y .* ((p ./ P) .^ (- ))
    l = (y ./ A) .^ (1 ./  )
    pi = p .* y - w .* l .- f0
    pr = y ./ l
    y[findall(x -> x < 0, pi)] .= 0
    l[findall(x -> x < 0, pi)] .= 0
    return y, l, pr, pi
end
```

[73]:
```
firm (generic function with 1 method)
```

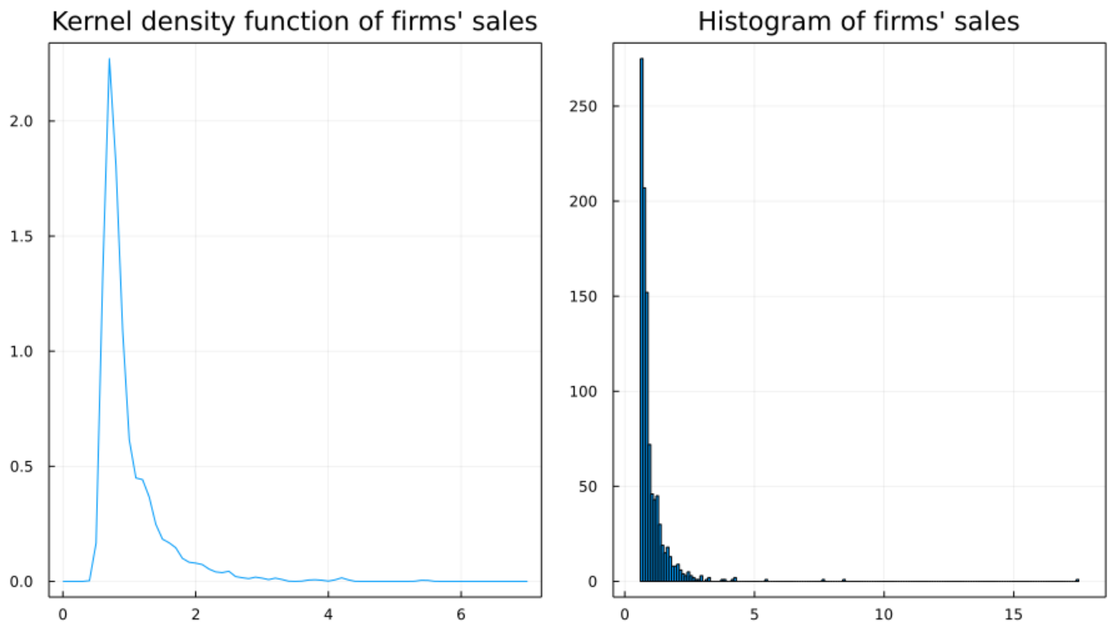
1.5

[79]:
```
memory = firm(3.5, 1, 2.4, 1, 100, 0.5, 1.2, 4);
```

1.6

[122]:
```
gr(fmt = :png, size = (900, 500))
plots = []
p1 = plot(0:0.1:7, pdf(kde(memory[1]), 0:0.1:7), title = "Kernel density␣
  ↪function of firms' sales")
push!(plots, p1)
p2 = plot(histogram(memory[1]), title = "Histogram of firms' sales")
push!(plots, p2)
plot(plots..., legend=false, framestyle = :box)
```

[122]:



1.7

As the number of samples increases, the power of the estimation increases and with about 10^5 samples, we have a good estimate

[108]:
```
# The moments that used in each part have a value of 1 otherwise 0
function SMM(list;  =3.5, f0=1.2, q=0.8, dist=1)
    A_bar=1
     =2.4
    P=1
    Y=100
    w=4
    R=100000
    N=1000
    Eg1 = mean(memory[1])
    Eg2 = mean(memory[2])
    Eg3 = mean(memory[3])
    Eg4 = mean(w ./ ((memory[4] ./ memory[2]) .- w))
    Eg5 = std(memory[3])
    Eg6 = std(memory[2])
    if dist == 1
        Aa = rand(Pareto( ,A_bar),R)
    else
        Aa = rand(LogNormal(A_bar, ),R)
    end
    p( ) = ((  ./ ( -1)) .* (w ./  ) .* ((1 ./ Aa) .^ (1 ./  )) .* (Y .^ ((1- ) ./␣
↪)) .* (P .^ ( *(1- ) ./  ))) .^ (  ./ ( - *( -1)))
```

```
    y( ) = Y .* ((p( ) ./ P) .^ (- ))
    l( ) = (y( ) ./ Aa) .^ (1 ./  )
    pr( ) = y( ) ./ l( )
    pi( ) = p( ) .* y( ) - w .* l( ) .- f0
    m1( ) = mean(y( )) .- Eg1
    m2( ) = mean(l( )) .- Eg2
    m3( ) = mean(pr( )) .- Eg3
    m4( ) = mean(w ./ ((pi( ) ./ l( )) .- w)) .- Eg4
    m5( ) = std(pr( )) .- Eg5
    m6( ) = std(l( )) .- Eg6
    JJ( ) = list[1].*m1( )^2 + list[2].*m2( )^2 + list[3].*m3( )^2 + list[4].
↪*m4( )^2 + list[5].*m5( )^2 + list[6].*m6( )^2
    opt = optimize(JJ, 0.01, 1)
    Alpha = opt.minimizer
    j = N*JJ(Alpha)
    rk = sum(list)-1
    chi = cquantile(Chisq(rk), q)
    println("Estimated   = $Alpha")
    println("Sargan-Hansen J-test:")
    if j < chi
        println("Chi-Square($rk, $q) = $(chi) and $(j) < $(chi) so the model␣
↪doesn't become rejected")
    else
        println("Chi-Square($rk, $q) = $(chi) and $(j) > $(chi) so the model␣
↪become rejected")
    end
end
```

[108]:
```
SMM (generic function with 1 method)
```

1.7.1 a

[83]:
```
list = [1 ,0, 1, 0, 0, 0]
x = SMM(list)
```

```
Estimated   = 0.500111428358997
Sargan-Hansen J-test:
Chi-Square(1, 0.8) = 0.06418475466730157 and 0.03621236690547079 <
0.06418475466730157 so the model doesn't become rejected
```

1.7.2 b

[82]:
```
list = [1 ,0, 1, 0, 1, 0]
x = SMM(list)
```

```
Estimated   = 0.5002995111070118
Sargan-Hansen J-test:
```

```
Chi-Square(2, 0.8) = 0.4462871026284194 and 0.15597531700723316 <
0.4462871026284194 so the model doesn't become rejected
```

1.7.3 c

[85]:
```python
list = [0 ,1, 0, 1, 0, 0]
x = SMM(list)
```

```
Estimated   = 0.4997936663455308
Sargan-Hansen J-test:
Chi-Square(1, 0.8) = 0.06418475466730157 and 0.00044872592186630543 <
0.06418475466730157 so the model doesn't become rejected
```

1.7.4 d

[86]:
```python
list = [0 ,1, 0, 1, 0, 1]
x = SMM(list)
```

```
Estimated   = 0.4998047117545865
Sargan-Hansen J-test:
Chi-Square(2, 0.8) = 0.4462871026284194 and 0.02781792050811224 <
0.4462871026284194 so the model doesn't become rejected
```

1.7.5 e

[87]:
```python
list = [1 ,1, 1, 1, 0, 0]
x = SMM(list)
```

```
Estimated   = 0.4998647211218645
Sargan-Hansen J-test:
Chi-Square(3, 0.8) = 1.005174013052349 and 0.15670453374752538 <
1.005174013052349 so the model doesn't become rejected
```

1.7.6 f

[88]:
```python
list = [1 ,1, 1, 1, 1, 1]
x = SMM(list)
```

```
Estimated   = 0.49984985312588537
Sargan-Hansen J-test:
Chi-Square(5, 0.8) = 2.3425343058411205 and 0.5500094063918348 <
2.3425343058411205 so the model doesn't become rejected
```

1.8

[94]:
```python
#Estimation
 =3.5
A_bar=1
 =2.4
```

```
P=1
Y=100
f0=1.2
w=4
R=100000
N=1000
Eg1 = mean(memory[2])
Eg2 = mean(w ./ ((memory[4] ./ memory[2]) .- w))
Aa = rand(Pareto( ,A_bar),R)
p( ) = ((  ./ ( -1)) .* (w ./  ) .* ((1 ./ Aa) .^ (1 ./  )) .* (Y .^ ((1- ) ./  ))␣
  ↪.* (P .^ ( *(1- ) ./  ))) .^ (  ./ ( - *( -1)))
y( ) = Y .* ((p( ) ./ P) .^ (- ))
l( ) = (y( ) ./ Aa) .^ (1 ./  )
pr( ) = y( ) ./ l( )
pi( ) = p( ) .* y( ) - w .* l( ) .- f0
m1( ) = mean(l( )) .- Eg1
m2( ) = mean(w ./ ((pi( ) ./ l( )) .- w)) .- Eg2
J( ) = m1( )^2 + m2( )^2
opt = optimize(J, 0.01, 1)
alpha = opt.minimizer

#Verification
Eg1 = mean(memory[1])
Eg2 = mean(memory[3])
m1( ) = mean(y( )) .- Eg1
m2( ) = mean(pr( )) .- Eg2
J( ) = m1( )^2 + m2( )^2
j = N*J(alpha)
chi = cquantile(Chisq(1), .8)
println("Estimated   = $alpha")
println("Sargan-Hansen J-test:")
println("Chi-Square(1, .80) = $(chi) and $(j) < $(chi) so the model doesn't␣
  ↪become rejected")
```

```
Estimated   = 0.4997346805302937
Sargan-Hansen J-test:
Chi-Square(1, .80) = 0.06418475466730157 and 0.04137641214763564 <
0.06418475466730157 so the model doesn't become rejected
```

1.9

1.9.1 =3

[100]:
```
#a
println(" =3, Part_a:")
list = [1 ,0, 1, 0, 0, 0]
x = SMM(list,  =3, q=0.1)
```

```
#b
println(" =3, Part_b:")
list = [1 ,0, 1, 0, 1, 0]
x = SMM(list,  =3, q=0.1)
```

```
 =3, Part_a:
Estimated   = 0.5091538457376763
Sargan-Hansen J-test:
Chi-Square(1, 0.1) = 2.7055434540954155 and 23.198823215125532 >
2.7055434540954155 so the model become rejected
 =3, Part_b:
Estimated   = 0.5116881015979313
Sargan-Hansen J-test:
Chi-Square(2, 0.1) = 4.605170185988092 and 37.02313184985233 > 4.605170185988092
so the model become rejected
```

1.9.2 f0=1.8

[101]:
```
#a
println("f0=1.8, Part_a:")
list = [1 ,0, 1, 0, 0, 0]
x = SMM(list, f0=1.8, q=0.1)

#b
println("f0=1.8, Part_b:")
list = [1 ,0, 1, 0, 1, 0]
x = SMM(list, f0=1.8, q=0.1)
```

```
f0=1.8, Part_a:
Estimated   = 0.5000282189588827
Sargan-Hansen J-test:
Chi-Square(1, 0.1) = 2.7055434540954155 and 0.062109389969983685 <
2.7055434540954155 so the model doesn't become rejected
f0=1.8, Part_b:
Estimated   = 0.5007488508531529
Sargan-Hansen J-test:
Chi-Square(2, 0.1) = 4.605170185988092 and 0.574885042907582 < 4.605170185988092
so the model doesn't become rejected
```

1.10 Optional

1.11

[110]:
```
#a
println("A~LogNormal, Part_a:")
list = [1 ,0, 1, 0, 0, 0]
x = SMM(list, q=0.1, dist=0)
```

```
#b
println("A~LogNormal, Part_b:")
list = [1 ,0, 1, 0, 1, 0]
x = SMM(list, q=0.1, dist=0)
```

```
A~LogNormal, Part_a:
Estimated   = 0.026690554370245704
Sargan-Hansen J-test:
Chi-Square(1, 0.1) = 2.7055434540954155 and 1.404641225144115e9 >
2.7055434540954155 so the model become rejected
A~LogNormal, Part_b:
Estimated   = 0.027738614329287425
Sargan-Hansen J-test:
Chi-Square(2, 0.1) = 4.605170185988092 and 1.6153504952171986e9 >
4.605170185988092 so the model become rejected
```

1.12

[113]:
```
function SMM2(list)
    A_bar=1
     =2.4
    P=1
    Y=100
    w=4
    R=1000000
    N=1000
    Eg1 = mean(memory[1])
    Eg2 = mean(memory[2])
    Eg3 = mean(memory[3])
    Eg4 = mean(w ./ ((memory[4] ./ memory[2]) .- w))
    Eg5 = std(memory[3])
    Eg6 = std(memory[2])
    Eg7 = std(memory[4])
    Eg8 = mean(memory[4])
    A( ) = rand(Pareto( [2],A_bar),R)
    p( ) = ((  ./ ( -1)) .* (w ./  [1]) .* ((1 ./ A( )) .^ (1 ./  [1])) .* (Y .^␣
↪((1- [1]) ./  [1])) .* (P .^ ( *(1- [1]) ./  [1]))) .^ ( [1] ./ ( [1]- *( [1]-1)))
    y( ) = Y .* ((p( ) ./ P) .^ (- ))
    l( ) = (y( ) ./ A( )) .^ (1 ./  [1])
    pr( ) = y( ) ./ l( )
    pi( ) = p( ) .* y( ) - w .* l( ) .-  [3]
    m1( ) = mean(y( )) .- Eg1
    m2( ) = mean(l( )) .- Eg2
    m3( ) = mean(pr( )) .- Eg3
    m4( ) = mean(w ./ ((pi( ) ./ l( )) .- w)) .- Eg4
    m5( ) = std(pr( )) .- Eg5
    m6( ) = std(l( )) .- Eg6
    m7( ) = std(pi( )) .- Eg7
    m8( ) = mean(pi( )) .- Eg8
    J( ) = list[1].*m1( )^2 + list[2].*m2( )^2 +  list[3].*m3( )^2 + list[4].
 ↪*m4( )^2 + list[5].*m5( )^2 + list[6].*m6( )^2 + list[7].*m7( )^2 + list[8].
 ↪*m8( )^2
    res = bboptimize(J, SearchRange = [(0.2,0.6), (3,3.8), (1,2)],␣
 ↪NumDimensions = 3, MaxTime = 30);
    alpha = best_candidate(res)[1]
    theta = best_candidate(res)[2]
    F0 = best_candidate(res)[3]
    println("Estimated   = $alpha")
    println("Estimated   = $theta")
    println("Estimated f0 = $F0")
end
```

[113]: SMM2 (generic function with 1 method)

1.12.1 7_d

[114]:
```
list = [0 ,1, 0, 1, 0, 1, 0, 0]
x = SMM2(list)
```

```
Starting optimization with optimizer DiffEvoOpt{FitPopulation{Float64},
RadiusLimitedSelector, BlackBoxOptim.AdaptiveDiffEvoRandBin{3},
RandomBound{ContinuousRectSearchSpace}}
0.00 secs, 0 evals, 0 steps
6.42 secs, 2 evals, 1 steps, fitness=8519.802001140
13.02 secs, 4 evals, 2 steps, fitness=5663.280427052
19.51 secs, 6 evals, 3 steps, improv/step: 0.333 (last = 1.0000),
fitness=5663.280427052
25.77 secs, 8 evals, 4 steps, improv/step: 0.250 (last = 0.0000),
fitness=353.283555342

Optimization stopped after 5 steps and 32.02 seconds
Termination reason: Max time (30.0 s) reached
Steps per second = 0.16
Function evals per second = 0.31
Improvements/step = Inf
Total function evaluations = 10


Best candidate found: [0.4917, 3.55902, 1.89449]

Fitness: 353.283555342

Estimated   = 0.49169989396906977
Estimated   = 3.559015492721119
Estimated f0 = 1.8944908474778375
```

1.12.2 7_e

[121]:
```
list = [1 ,1, 1, 1, 0, 0, 0, 0]
x = SMM2(list)
```

```
Starting optimization with optimizer DiffEvoOpt{FitPopulation{Float64},
RadiusLimitedSelector, BlackBoxOptim.AdaptiveDiffEvoRandBin{3},
RandomBound{ContinuousRectSearchSpace}}
0.00 secs, 0 evals, 0 steps
6.26 secs, 2 evals, 1 steps, fitness=15.997522025
12.54 secs, 4 evals, 2 steps, improv/step: 0.500 (last = 1.0000),
fitness=6.383858267
18.86 secs, 6 evals, 3 steps, improv/step: 0.333 (last = 0.0000),
fitness=2.793660889
25.05 secs, 8 evals, 4 steps, improv/step: 0.250 (last = 0.0000),
fitness=2.793660889

Optimization stopped after 5 steps and 31.25 seconds
Termination reason: Max time (30.0 s) reached
Steps per second = 0.16
Function evals per second = 0.32
Improvements/step = Inf
Total function evaluations = 10


Best candidate found: [0.560274, 3.52187, 1.77579]

Fitness: 2.150121860

Estimated   = 0.5602740527827594
Estimated   = 3.5218730470704296
Estimated f0 = 1.7757860753103878
```

1.12.3 7_f

[120]:
```
list = [1 ,1, 1, 1, 1, 1, 0, 0]
x = SMM2(list)
```

```
Starting optimization with optimizer DiffEvoOpt{FitPopulation{Float64},
RadiusLimitedSelector, BlackBoxOptim.AdaptiveDiffEvoRandBin{3},
RandomBound{ContinuousRectSearchSpace}}
0.00 secs, 0 evals, 0 steps
6.32 secs, 2 evals, 1 steps, fitness=187.659612468
12.53 secs, 4 evals, 2 steps, improv/step: 0.500 (last = 1.0000),
fitness=187.659612468
18.81 secs, 6 evals, 3 steps, improv/step: 0.333 (last = 0.0000),
fitness=187.659612468
25.25 secs, 8 evals, 4 steps, improv/step: 0.500 (last = 1.0000),
fitness=187.659612468

Optimization stopped after 5 steps and 31.84 seconds
Termination reason: Max time (30.0 s) reached
Steps per second = 0.16
Function evals per second = 0.31
Improvements/step = Inf
Total function evaluations = 10


Best candidate found: [0.588065, 3.78819, 1.17879]

Fitness: 187.659612468

Estimated   = 0.5880652763155233
Estimated   = 3.7881859270954235
Estimated f0 = 1.1787890194702726
```

1.12.4 7_f + 2 more moments of mean and std of profit

[79]:
```
list = [1 ,1, 1, 1, 1, 1, 1, 1]
x = SMM2(list)
```

```
Starting optimization with optimizer DiffEvoOpt{FitPopulation{Float64},
RadiusLimitedSelector, BlackBoxOptim.AdaptiveDiffEvoRandBin{3},
RandomBound{ContinuousRectSearchSpace}}
0.00 secs, 0 evals, 0 steps
6.43 secs, 2 evals, 1 steps, fitness=97080.762848891
12.79 secs, 4 evals, 2 steps, improv/step: 0.500 (last = 1.0000),
fitness=97080.762848891
19.15 secs, 6 evals, 3 steps, improv/step: 0.667 (last = 1.0000),
fitness=97080.762848891
25.68 secs, 8 evals, 4 steps, improv/step: 0.750 (last = 1.0000),
fitness=5483.129907093

Optimization stopped after 5 steps and 32.26 seconds
Termination reason: Max time (30.0 s) reached
Steps per second = 0.16
Function evals per second = 0.31
Improvements/step = Inf
Total function evaluations = 10


Best candidate found: [0.486931, 3.73057, 1.18443]

Fitness: 5483.129907093
```

```
Estimated   = 0.4869305584189828
Estimated   = 3.7305712101915898
Estimated f0 = 1.1844318496137993
```

1.12.5 9

We want to estimate θ and f_0, so it doesn't make sense to consider them as the default model parameters!!!