John Higgins
Programming Bootcamp
27 June, 2022

Problem 1.

Solution: Code (and output) for problem 1 is included below:

```
##problem 1
function facto(n)
    y = 1
    if n >= 0 && isinteger(n)
        for i in 1:n
            y = y*i
        end
        return y
    else
        println("Invalid input! Needs to be non-negative integer")
    end
end

println(facto(3))

>> 6
```

□

Problem 2.

Solution: Code (and output) for problem 2 is included below:

```
function p(coeff,x)
    y = 0
    for (i,a_i) in enumerate(coeff)
        y += a_i*x^(i-1)
    end
    return y
end

coeff = [1,2,3]
```

```
x = 2
println(p(coeff,x))
>>17
```

□

Problem 3.

Solution: Code (and output) for problem 3 is included below:

```
function sq(x)
    y = 0
    for (index, value) in enumerate(x)
        y += value^2
    end
    return y
end

function approx_n(n)
    succ = 0
    for i in 0:n
        x = rand(2,1)
        if sq(x) <=1
            succ +=1
        end
    end
    return 4*succ/n
end

n_range = [100, 1000, 10000, 100000,1000000,10000000]
for n in n_range
    println(String("$(n): $(approx_n(n))"))
end

>>100: 3.16
```

```
>>1000: 3.144
>>10000: 3.1508
>>100000: 3.13212
>>1000000: 3.139208
>>10000000: 3.1423068
```

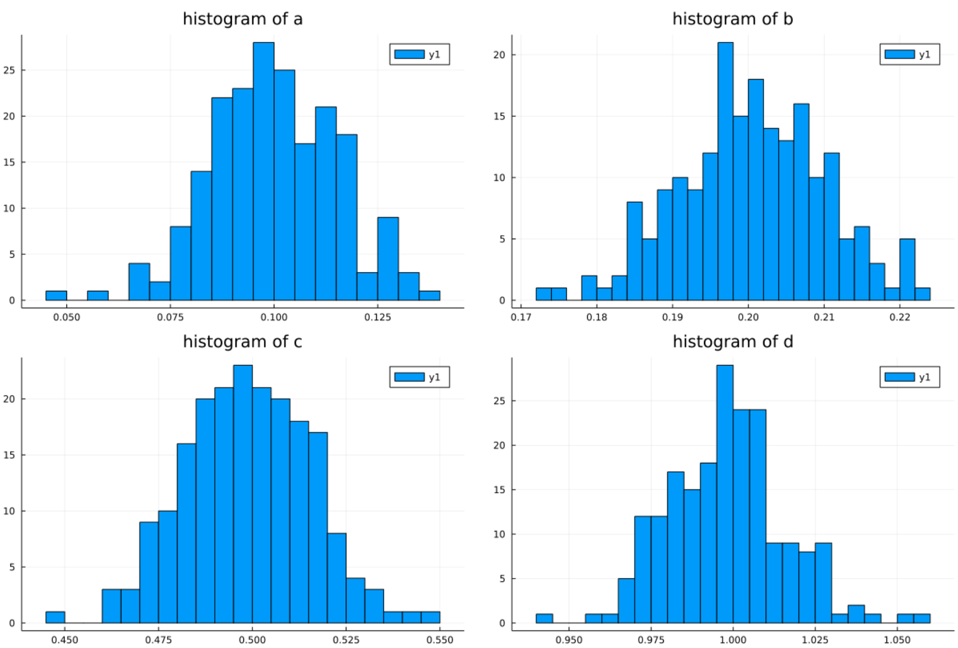
□

Problem 4.

Solution: Code (and output) for problem 4 is included below:

```
n = 50
w_len = 200
x1 = randn(n)
x12 = x1.^2
x2 = randn(n)
X = hcat(x1, x12, x2, ones(n))
coeff = zeros(4,w_len)
for i in 1:w_len
    w = randn(n)
    Y = 0.1.*x1 + 0.2.*x12 + 0.5.*x2 + ones(n) + 0.1.*w
    \beta = inv(X'*X)*X'*Y
    pred = X*\beta
    for j in 1:4
        coeff[j,i] = \beta[j]
    end
end

using Plots
using Distributions
hista = histogram(coeff[1,:], bins=30, title="histogram of a")
histb= histogram(coeff[2,:], bins=30, title="histogram of b")
histc= histogram(coeff[3,:], bins=30, title="histogram of c")
histd = histogram(coeff[4,:], bins=30, title="histogram of d")
```

```
savefig(hista, "hista.png")
savefig(histb, "histb.png")
savefig(histc, "histc.png")
savefig(histd, "histd.png")
```

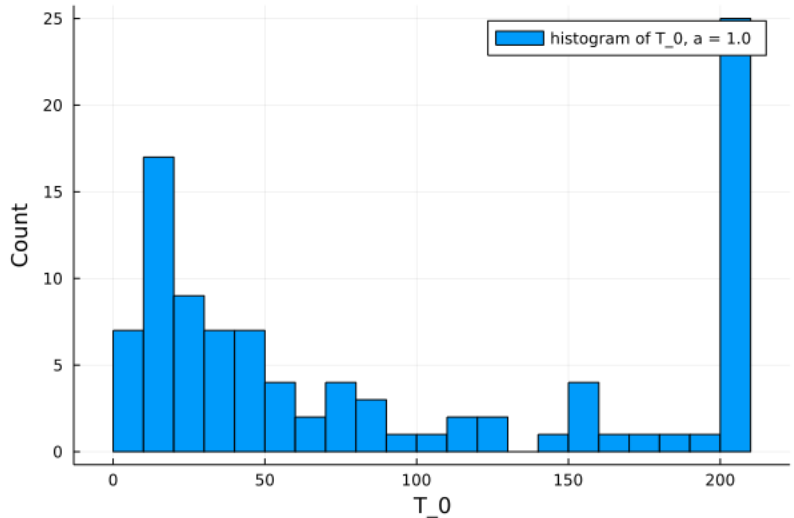


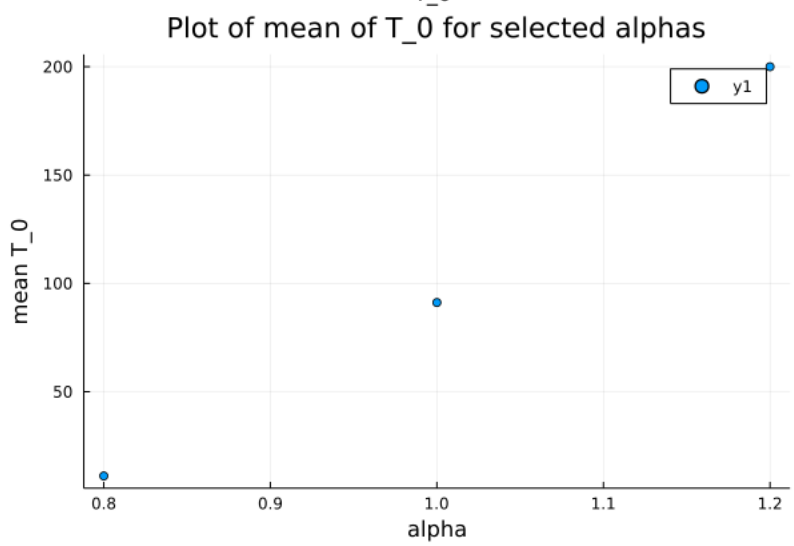
Problem 5.

Solution: Code (and output) for problem 5 is included below:

```
function rw_step(x, t, a)
    \eps = randn()
    if t < 200
        x1 = a*x + 0.2*\eps
    else
        x1 = 0
    end
    return x1
end

```

```julia
function first_pass(x,t,a)
    while x > 0
        t +=1
        x = rw_step(x,t,a)
    end
    return t
end

function sample_collect(a)
    hit = zeros(100)
    for i in 1:100
        x=1
        t=0
        hit[i] = first_pass(x,t,a)
    end
    return hit
end

using Statistics
a_list = [0.8, 1, 1.2]
a_mean = zeros(3)
for (index, a) in enumerate(a_list)
    a_hit = sample_collect(a)
    hist_a = histogram(a_hit, bins=30, label="histogram of T_0, a = $(a)", xlabel="T_0", ylabel = "Count")
    savefig(hist_a, "t0 hist, a = $(a).png")
    a_mean[index] = mean(a_hit)
end
mean_plot = plot(a_list, a_mean, seriestype = :scatter, title="Plot of mean of T_0 for selected alphas", xlabel="alpha", ylabel="mean T_0")
savefig(mean_plot, "mean_plot.png")
```





□

Problem 6.

Solution: Code (and output) for problem 6 is included below:

```
function f_x(x)
    return (x-1)^3
end
function f_pr(x)
    return 3*(x-1)^2
end

```

```
function f_x2(x)
    return 10 + x - x^2
end

function f_pr2(x)
    return 1-2x
end

function root_finder(f, f_prime, x_0, tol, maxiter)
    x = x_0
    t = 1
    while (f(x))/(f_prime(x)) > tol && t <= maxiter
        x = x - (f(x))/(f_prime(x))
        println(x)
        t+=1
    end
    return x
end

root_finder(f_x, f_pr, 2, 0.0001, 1000)
>>1.0002
root_finder(f_x2, f_pr2, 8, 0.0001, 10000)
>>3.7015
```

□

Problem 7.

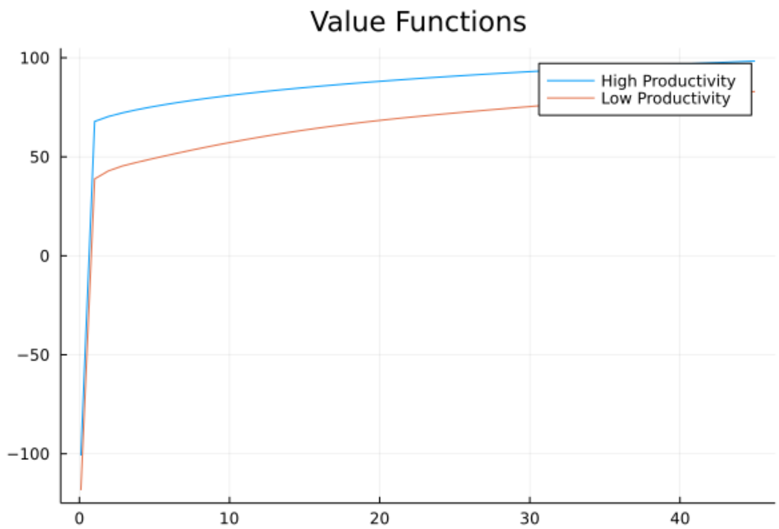
Solution: Code (and output) for problem 7 is included below:

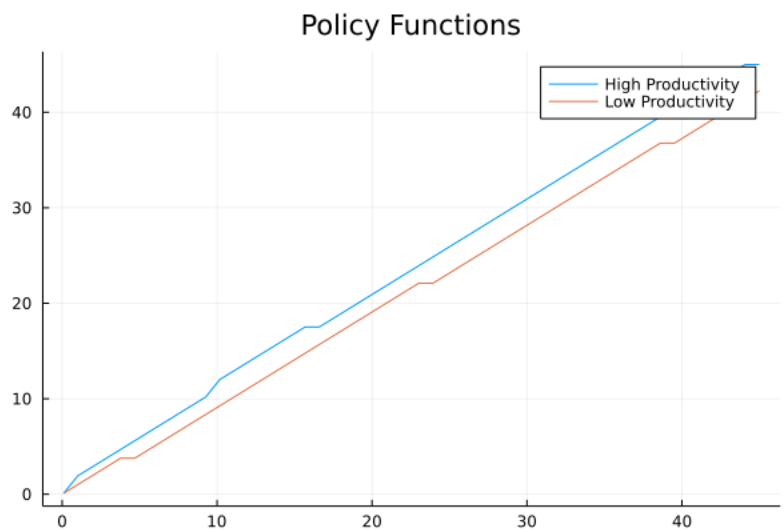
```
using Parameters, Plots #read in necessary packages

@with_kw struct Primitives
    \beta::Float64 = 0.99 #discount factor
    \theta::Float64 = 0.36 #production
```

```
    \delta::Float64 = 0.025 #depreciation
    k_grid::Array{Float64,1} = collect(range(0.1, length = 50, stop= 45.0)) #capital grid
    nk::Int64 = length(k_grid) #number of capital grid states
    Pi::Array{Float64,2} = [0.977  0.023; 0.074  0.926] #initialize productivity transition grid
    prod_mat::Array{Float64,1} = [1.25, 0.2]#productivity matrix
    zk::Int64 = 2#length of productivity vector
end

mutable struct Results
    val_func::Array{Float64,2}  #value function
    pol_func::Array{Float64,2} #policy function
end

function solve_model()
    prim = Primitives()
    val_func, pol_func = zeros(prim.nk,2), zeros(prim.nk,2)#initialize policy function vectors
    res = Results(val_func, pol_func)#results

    error = 100
    n = 0
    tol = 1e-4
    while error>tol
        n+=1
        v_next=zeros(prim.nk,2)
        for i in 1:2
            v_next[:,i] .= Bellman(prim,res, i)
        end
        error = maximum(abs.(res.val_func .- v_next)) #reset error term
        res.val_func = v_next
        #update value function held in results vector
```

```
        println(n, "  ",  error)

        if mod(n, 5000) == 0 || error <tol
            println(" ")
            println("*************************************************")
            println("AT ITERATION = ", n)
            println("MAX DIFFERENCE = ", error)
            println("*************************************************")
        end
    end
    prim,res
    println("Value function converged in ", n, " iterations.")
    vfplot = Plots.plot(prim.k_grid, res.val_func, title="Value Functions", labels=["High Productivity" "Low Productivity"]) #plot value function
    pfplot = Plots.plot(prim.k_grid, res.pol_func, title="Policy Functions", labels=["High Productivity" "Low Productivity"]) #plot value function
    display(vfplot)
    display(pfplot)
    savefig(vfplot, "vfplot.png")
    savefig(pfplot, "pfplot.png")
end

#Bellman operator
function Bellman(prim::Primitives, res::Results, prod_ind)
    @unpack \beta,\delta,\theta,nk, k_grid, Pi, prod_mat,zk = prim
    v_next = zeros(nk,2)

    for i_k = 1:nk #loop over state space
        candidate_max = -1e10 #something crappy
        k = k_grid[i_k]#convert state indices to state values
        prod = prod_mat[prod_ind] #convert current productivity index to productivity value
```

```
        budget = prod*k^{\theta} + (1-\delta)*k #budget given current
   state. Doesn't this look nice?

        for i_kp = 1:nk #loop over choice of k_prime
            kp = k_grid[i_kp]
            c = budget - kp #consumption
            if c>0 #check to make sure that consumption is positive
                val = log(c)
                for i in 1:zk
                    val += \beta*Pi[prod_ind,i]*res.val_func[i_kp, i]
                end
                if val>candidate_max #check for new max value
                    candidate_max = val
                    res.pol_func[i_k,prod_ind] = kp #update policy
   function
                end
            end
        end
        v_next[i_k,prod_ind] = candidate_max #update next guess of value
   function
    end
    v_next[:,prod_ind]
end

solve_model() #solve the model.

using Profile

Profile.clear()
@profile solve_model()
Profile.print()
#############
```


Value Functions
High Productivity
Low Productivity
100
50
0
−50
−100
0
10
20
30
40


Policy Functions
High Productivity
Low Productivity
40
30
20
10
0
0
10
20
30
40

□