In [2]:
```
##Define Parameters
α =  0.36; # Capital share in production function
β = 0.99; # Discount factor
δ = 0.025; # Depreciation rate
```

In [4]:
```
##Utility function
## TO FILL u(c)
function u(c)
    if c > 0
        return log(c)
    else
        return 1e-9
    end
end;
##Production function
## TO FILL F(k)
function F(k)
    return k^α
end;
```

In [5]:
```
##Defining the steady state
k_ss = (α/((1/β) - (1-δ)))^(1/(1-α));  # Steady State Capital Stock
y_ss = F(k_ss);                 # Steady State Output
c_ss = y_ss - δ*k_ss;            # Steady State Consumption
```

In [6]:
```
##Defining the Grid for the Endogenous State Variable: Capital
nk = 501;                            # Number of Grid Points
kmin = 0.2*k_ss; kmax = 1.8*k_ss;    # Bounds for Grid
kg = kmin:(kmax-kmin)/(nk-1):kmax; # Equally Spaced Grid for Capital
```

In [21]:
```
## Build the 2-Dimensional Contemporaneous Utility Grid for the System
## TO FILL U: initialize the arra
U = Array{Float64}(undef, nk, nk)
for ii = 1:nk       # Loop Over Capital Today
    for jj = 1:nk   # Loop Over Capital Tomorrow
        k = kg[ii];     # Capital Today
        kp = kg[jj];    # Capital Tomorrow
        # Solve for Consumption at Each Point
        ## TO FILL
        c = F(k) + (1-δ)*k - kp
        if (kp < 0)||(c .< 0)
            # If Tomorrow"s Capital Stock | Today"s Consumption is Negative
```

```
            ## TO FILL
            U[ii,jj] = 1e-9
            # Numerical Trick to ensure that the Value Function is never
            # optimised at these points
        else
            ## TO FILL
            # Calculate Utility at this Point on Grid
            U[ii,jj] = u(c)
        end
    end
end
```

In [26]:

```
##Value Function Iteration

#Initial Guess of the Value Function
V0 = ones(nk,1) # nk x 1 vector of initial guess

tol = 1e-4;
maxits = 3000; # Define the maximum number of iterations
V_1 = V0;  # The new value function I obtain after an iteration
V_0 = V0;  # the  value function from which I start in each new iteration
dif = 1;
policy_k_index = Array{Int64,1}(undef,nk);

for iter in 1:maxits
    if dif < tol
        println("Converged at iteration: $iter")
        break
    else
        # TO FILL: find the fixed point of the Bellman equation ;)
        for ii in 1:nk
            vals = U[ii,:] .+ β .*V_0[:]
            V_1[ii], policy_k_index[ii] = findmax(vals)
        end
        dif = maximum(abs.(V_1 - V_0))
        V_0 =  copy(V_1)
    end
end
```

```
Converged at iteration: 2
```

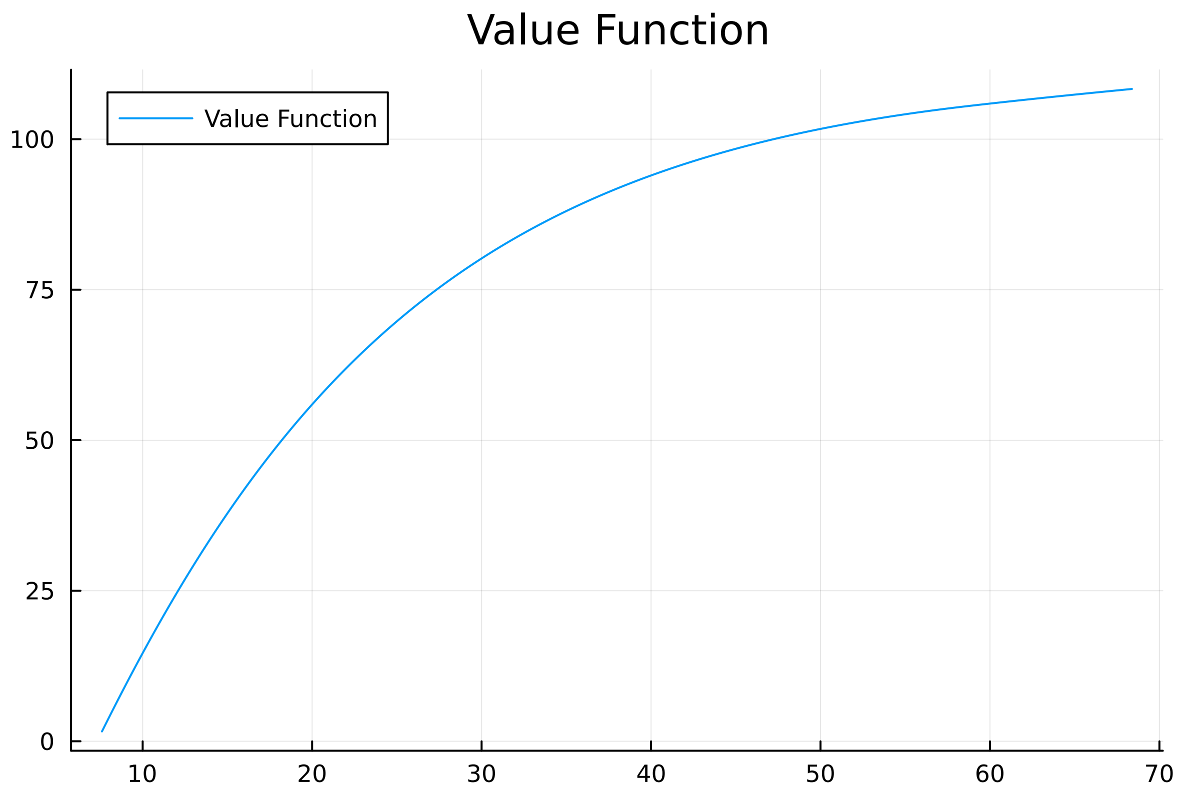
In [14]:

```
##Policy function for capital
policy_k = Array{Float64,1}(undef,nk);
policy_k = kg[policy_k_index]
```

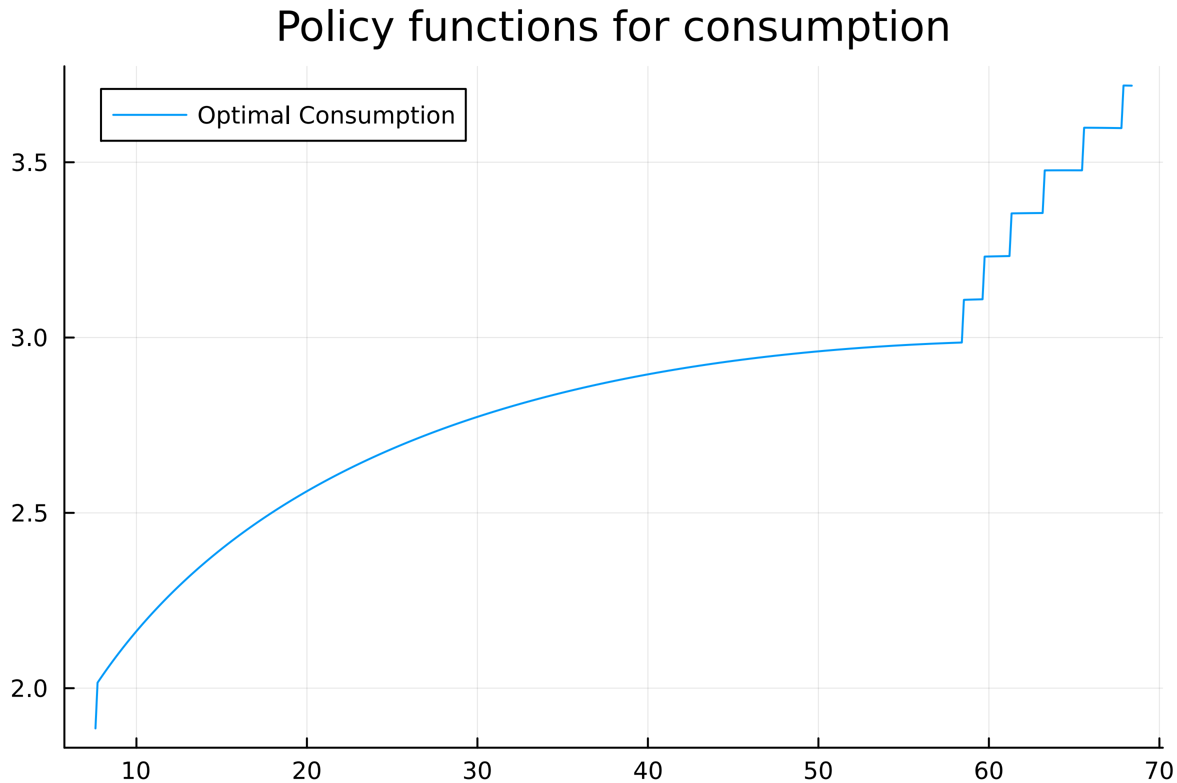
```julia
##Policy function for consumption
policy_c = Array{Float64,1}(undef,nk)

for k1 in 1:nk
    policy_c[k1] = F(kg[k1]) + (1-δ)*kg[k1] - policy_k[k1]
end
```

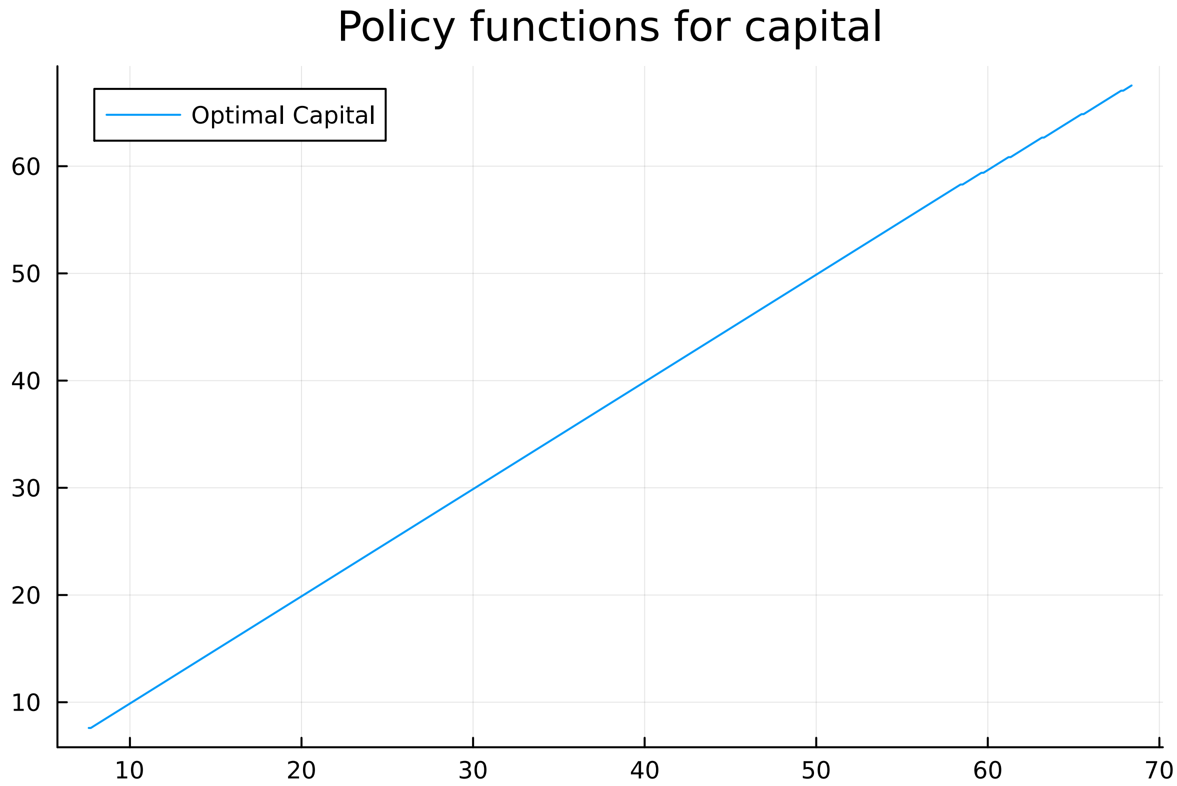
In [16]:
```julia
##Plotting the results
using Plots
plot(kg,V_1,title="Value Function",label="Value Function")
```



In [17]:
```julia
plot(kg,policy_c,title="Policy functions for consumption",label="Optimal Consumption")
```



In [18]:
```
plot(kg,policy_k,title="Policy functions for capital",label="Optimal Capital")
```


Policy functions for capital
Optimal Capital
10
20
30
40
50
60
10
20
30
40
50
60
70