solve-transition

March 21, 2019

```
In [1]: using Pkg
        pkg"activate ."
        pkg"instantiate"
        pkg"precompile"
```

```
  Updating registry at `~/.julia/registries/General`
  Updating git-repo `https://github.com/JuliaRegistries/General.git`
Precompiling project...
```

```
In [2]: using PerlaTonettiWaugh, Plots, BenchmarkTools, CSV
```

```
In [3]: gr(fmt = :png)
```

```
Out[3]: Plots.GRBackend()
```

1 Steady states

1.1 Solving steady state solutions

Set up parameters and find the corresponding stationary solution:

Below are the results from the **updated** matlab calibration with the corresponding letters. Here I added the μ parameter in the calibration routine to fit the firm dynamic moments. Fit improved a lot and it generated a negative drift term which is what we need to keep S from going negative.

```
In [4]: parameters = parameter_defaults()
```

```
Out[4]: ( = 0.02,  = 3.05, N = 10,  = 4.9411,  = 1.0,  = 0.1317,  = 1.0,  = 0.0, Theta = 1,  =
```

```
In [5]: #5.0672    3.2941    0.1152    4.7847   -0.0289    0.0819    2.3664

        parameters = merge(parameters, ( = 0.0195, d = 5.0672,  = 3.2941,   = 0.1152,  = 1/4.78
```

```
Out[5]: ( = 0.0195,  = 2.3664, N = 10,  = 3.2941,  = 1.0,  = 0.1152,  = 1.0,  = 0.0, Theta = 1
```

```
In [6]: changed_parameters = true # set this to true if you drastically change the parameters
```

```
Out[6]: true
```

Side note on previous version. Essentially, under this calibration, the parameters are very simmilar. Theta, chi, kappa are quite close to what we had in the prior version.

In [7]:
```
# Define common objects.
#parameters = parameter_defaults()

settings = settings_defaults()
settings = merge(settings, (transition_penalty_coefficient = 1.0, ))
z_grid = settings.z
P = length(z_grid)

d_0 = parameters.d # Here is the 10 percent tariff increase
d_T = 1 + (parameters.d-1)*0.90

d_autarky = 1 + (parameters.d-1)*2.5

params_0 = merge(parameters, (d = d_0, )) # parameters to be used at t = 0
params_T = merge(parameters, (d = d_T, )) # parameters to be used at t = T
params_autarky = merge(parameters, (d = d_autarky, )) # parameters to be used in autar

# initial value for numerical solver on (g, z_hat, Omega)
initial_x = [0.02; 2; .57]

# solve for stationary solution at t = 0
stationary_sol_0 = stationary_algebraic(params_0, initial_x) # solution at t = 0
stationary_sol = stationary_algebraic(params_T, initial_x) # solution at t = T
stationary_autarky = stationary_algebraic(params_autarky, initial_x) # solution at t =

_0 = stationary_sol_0.;
_T = stationary_sol.;
```

In [8]:
```
d_0
```

Out[8]:
```
5.0672
```

In [9]:
```
print(stationary_sol.U_bar,'\n')
print(stationary_sol_0.U_bar,'\n')
print(stationary_sol_0.S,'\n')
print(stationary_autarky.S,'\n')
```

```
31.960806397169257
25.335205621937796
0.08792930215911846
0.06111189985043804
```

In [10]:
```
#T = solved.t[end]
lambda_ss = 100*(consumption_equivalent(stationary_sol.U_bar, stationary_sol_0.U_bar,
```

```
        print("SS to SS welfare gain: ", lambda_ss,"\n")
        print("SS to SS welfare gain: ", stationary_sol._ii,"\n")
```

```
SS to SS welfare gain: 13.791679181400385
SS to SS welfare gain: 0.8650014500550756
```

```
In [11]: print("SS to SS welfare gain: ", stationary_sol.g,"\n")
```

```
SS to SS welfare gain: 0.011575084222623117
```

1.1.1 This is the autarky calculation

```
In [12]: #T = solved.t[end]
         lambda_ss_autarky = 100*(consumption_equivalent(stationary_autarky.U_bar, stationary_s

         print("SS to SS welfare gain: ", lambda_ss_autarky,"\n")
         print("Autarky Home Share, should be close to 1: ", stationary_autarky._ii,"\n")
         print("Autarky TFP growth rate ", stationary_autarky.g,"\n")
```

```
SS to SS welfare gain: -31.541980650737344
Autarky Home Share, should be close to 1: 0.9916620326433107
Autarky TFP growth rate 0.00069969517613008
```

1.2 Welfare in Steady States

1.2.1 Steady state at T

```
In [13]: stationary_sol.U_bar
```

```
Out[13]: 31.960806397169257
```

1.2.2 Steady state at 0

```
In [14]: stationary_sol_0.U_bar
```

```
Out[14]: 25.335205621937796
```

1.2.3 Outstanding Issue #1: Sensitivity of growth to trade.

This is a big difference relative to previous version. As noted above, with parameter values that are quite similar to what we had before, the growth rate is changing a lot with only a very small change in trade flows. Why?

```
In [15]: @show stationary_sol.g, stationary_sol_0.g;
```

```
(stationary_sol.g, stationary_sol_0.g) = (0.011575084222623117, 0.008840735918269178)
```

2 Transition dynamics

Setup for optimizer:

```
In [16]: settings = merge(settings, (params_T = params_T, stationary_sol_T = stationary_numeri
```

Use the solution found with calibrated parameters above for `E`

```
In [17]: settings = merge(settings, (transition_x0 = [-0.8738884967642274, -0.5125465450602664
                 tstops = (changed_parameters ? nothing : 1:settings.T))); # remove tstops whe
```

Find the corresponding solution

```
In [18]: @time result = solve_full_model(merge(settings, (transition_iterations = (changed_para
             impose_E_monotonicity_constraints = true, write_csv = true, csvpath = "solve-trans
             run_global = false)

         solved = result.solution;
         E_nodes = result.E_nodes;
         solved = solved.results;
```

```
261.967416 seconds (1.66 G allocations: 270.420 GiB, 13.19% gc time)
```

2.1 Welfare Gains

```
In [19]: print("Utility in initial SS: ", stationary_sol_0.U_bar,"\n")
         print("Utility in new SS: ", stationary_sol.U_bar,"\n")
```

```
Utility in initial SS: 25.335205621937796
Utility in new SS: 31.960806397169257
```

```
In [20]: print("Utility immediately after change ", solved.U[1],"\n")
```

```
Utility immediately after change 33.30189404780913
```

Summary so far... In the old paper, what we did was take U_0_ss at some date t, then compare it to U_ss at the same date t. This is like an instantaneous jump to the new ss. This is what the first cell is looking at and note that this is like a 17 percent increase in utility. Higher than what we had in the paper, but in of the same order of magnitude.

The next cell reprots the utility just after the change. Utility here includes the future path of consumption and change in growth rate, so it "bakes in" the transition path. Here it goes up by much more than utility in the new SS. This is what I was expecting given the dynamics of consumption.

Just a reminder about how the function `counsumption_equivalent(U_new, U_old, parameters)` works, it takes U_new and then U_old in that order, then evaluates the **gross** increase in consumption. 100 times this value **minus one** gives the permanent, percent increase in consumption required to make the agent indifferent between the two paths.

```
In [21]: #T = solved.t[end]
         lambda_ss = 100*(consumption_equivalent(stationary_sol.U_bar, stationary_sol_0.U_bar,

         print("SS to SS welfare gain: ", lambda_ss,"\n")
```

```
SS to SS welfare gain: 13.791679181400385
```

```
In [22]: lambda_tpath = 100*(consumption_equivalent(solved.U[1], stationary_sol_0.U_bar, parame

         print("Inclusive of the Transition Path: ", lambda_tpath,"\n")
```

```
Inclusive of the Transition Path: 16.806720853496373
```

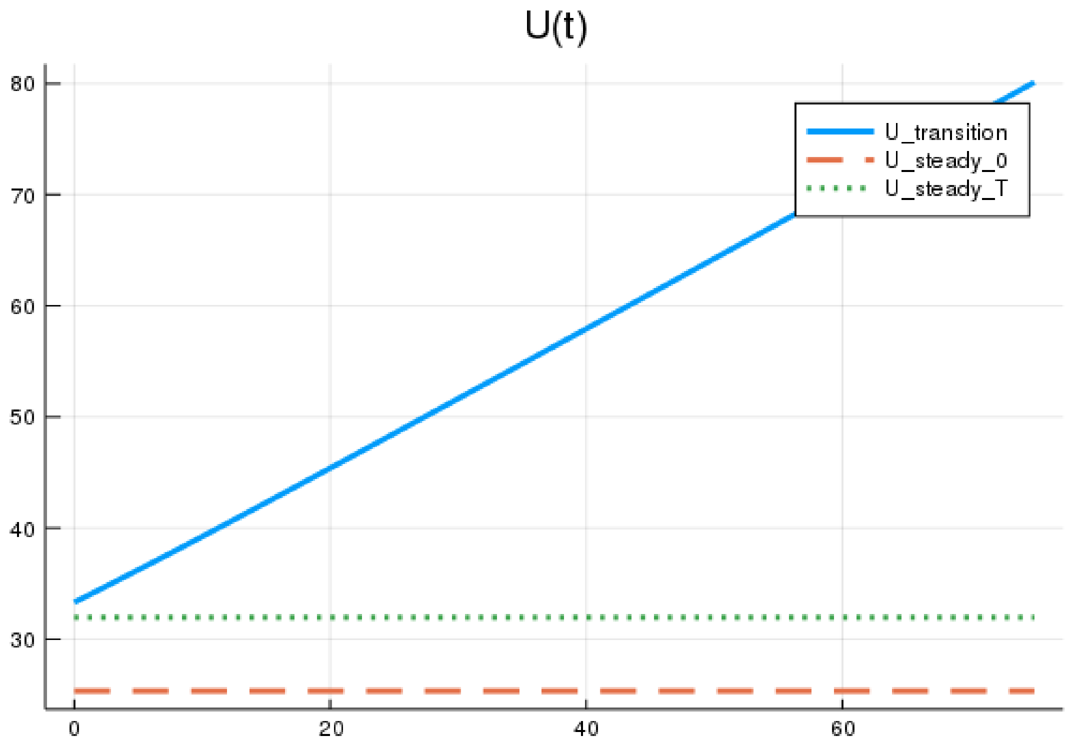
2.1.1 Relative to initial notebook computations

- SS to SS, the issue here is that we should compare **at date 0** utility in the first ss vs. the second ss. This is what we must have been doing in the previous version of the paper. The previous calulation in the old notebook had `consumption_equivalent(solved.U[end], stationary_sol_0.U_bar, parameters)` which took **date T** utility and compard them. The problem is that this now depends on date T. So if we picked T to be arbitraily large, then utility will be arbitraily different.
- Following the same logic, the transition path should compare **date 0** utility with the initial value from the transition path. So what we want to do is to compare everything at 0. In the previous calculation, we had `consumption_equivalent(solved.U[1], stationary_sol_0.U_bar, parameters)` were comparing the initial utility relative to ss utility on the old path at date T. So the initial blue point below versues the last orange dashed point. **See the figure below**

```
In [23]: # generate the plot!
         U_steady_0(t) = stationary_sol_0.U_bar
         U_steady_T(t) = stationary_sol.U_bar

         plot(solved.t,
             [solved.U, U_steady_0, U_steady_T],
             label = ["U_transition", "U_steady_0", "U_steady_T"] ,
             title = "U(t)", linestyle = :auto, lw = 3)
```

```
Out[23]:
```



2.1.2 Outstanding Issue #2: Welfare Gains still depend on T in transition path.

The stuff above I think is correct, the one issue is why does the welfare gains, inclusive of the transition path seem to depend on T??? So change T above from 40 to 75 or 100, then the welfare gains fall alot? Why?

2.2 Plotting

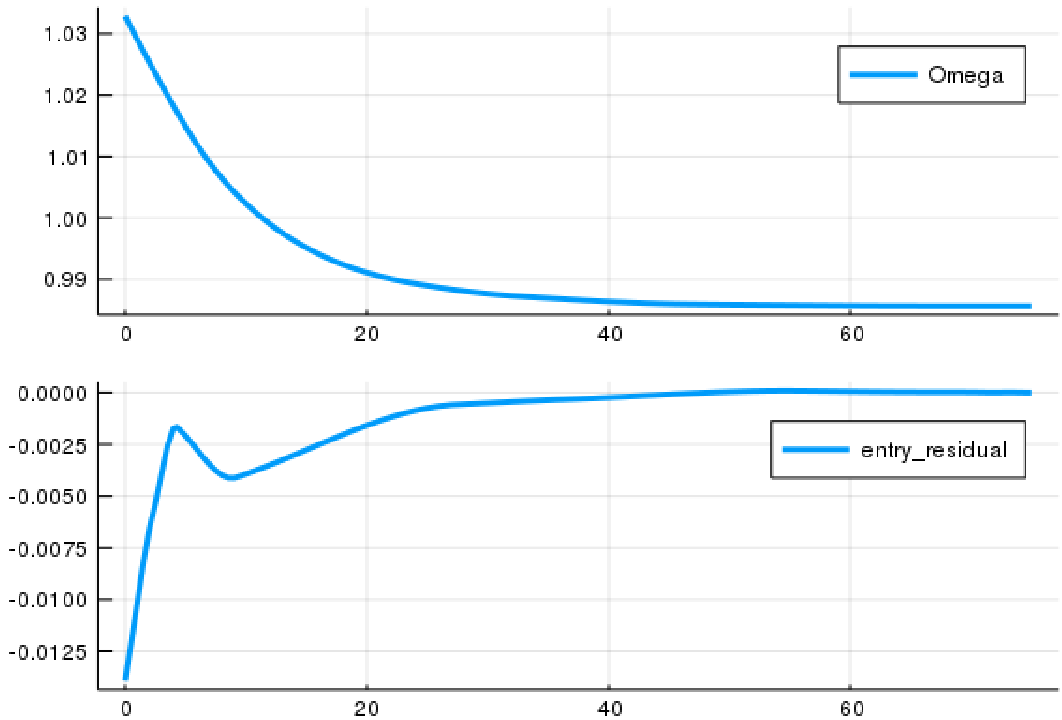
In [24]:
```
solved.U[end]
```

Out[24]:
```
80.11612246846532
```

2.3 Plots for and residuals

In [25]:
```
#solved = solved.results;
plot_ = plot(solved.t, solved., label = "Omega", lw = 3)
plot_residual = plot(solved.t, solved.entry_residual, label = "entry_residual", lw = 
plot(plot_, plot_residual, layout = (2,1))
```

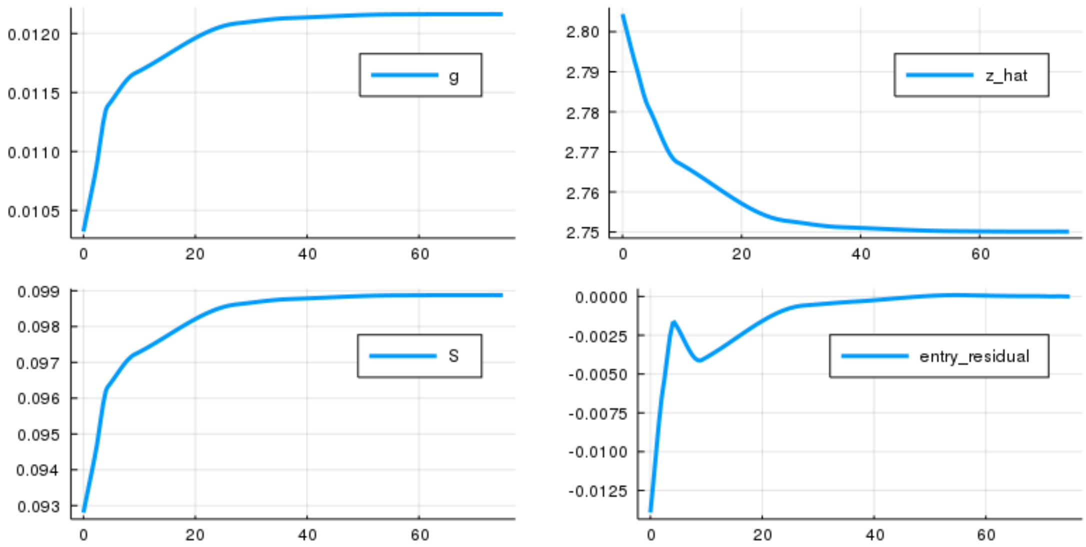
Out[25]:



2.4 Primary Plots

In [26]:
```
plot1 = plot(solved.t, solved.g, label = "g", lw = 3)
plot2 = plot(solved.t, solved.z_hat, label = "z_hat", lw = 3)
plot3 = plot(solved.t, solved.S, label = "S", lw = 3)
plot4 = plot(solved.t, solved.entry_residual, label = "entry_residual", lw = 3)
plot(plot1, plot2, plot3, plot4, layout=(2,2), size = (800, 400))
```

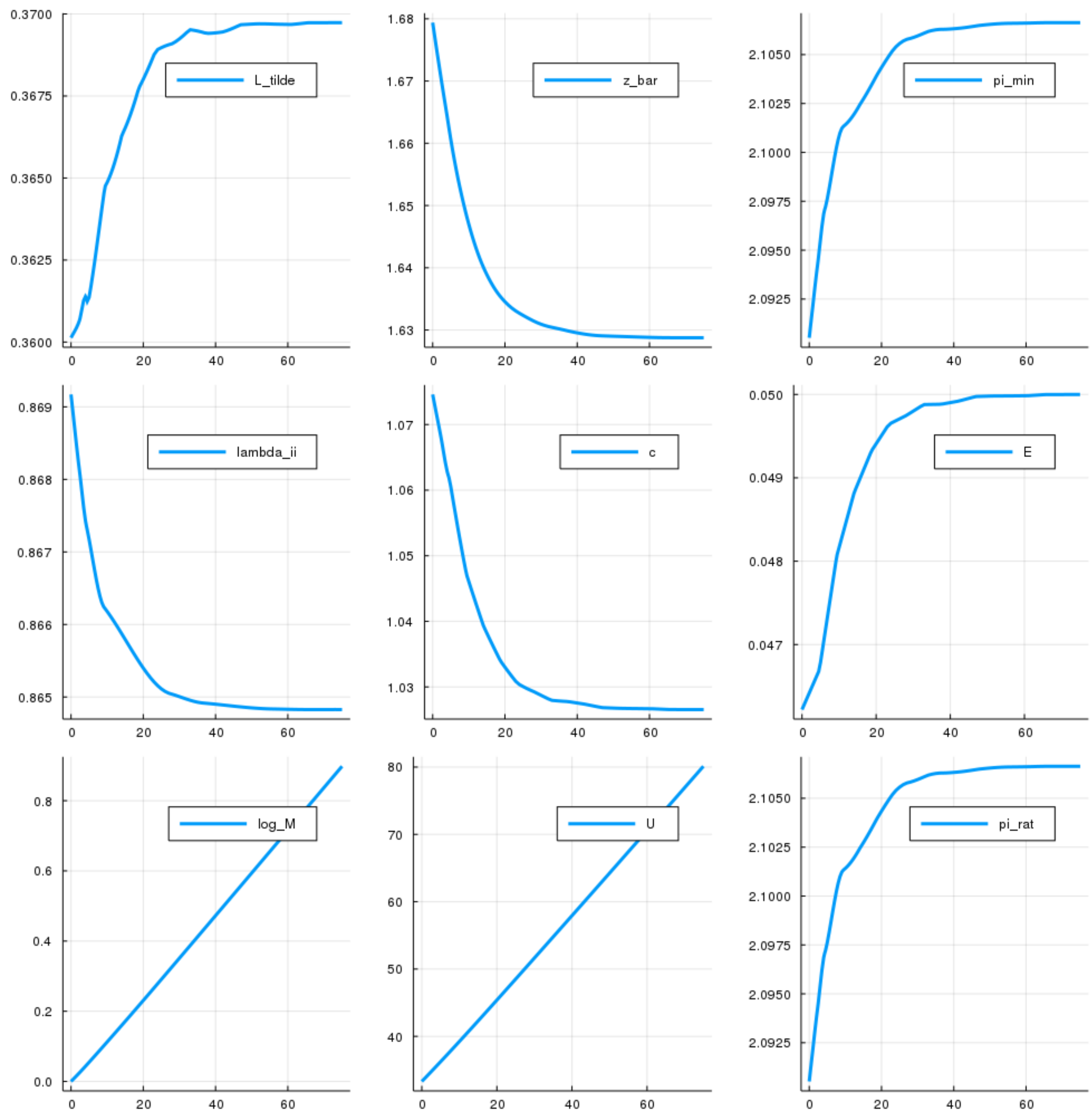
Out[26]:



2.5 Static Equations

In [27]:
```
plot1 = plot(solved.t, solved.L_tilde, label = "L_tilde", lw = 3)
plot2 = plot(solved.t, solved.z_bar, label = "z_bar", lw = 3)
plot3 = plot(solved.t, solved._rat, label = "pi_min", lw = 3)
plot4 = plot(solved.t, solved._ii, label = "lambda_ii", lw = 3)
plot5 = plot(solved.t, solved.c, label = "c", lw = 3)
plot6 = plot(solved.t, solved.E, label = "E", lw = 3)
plot7 = plot(solved.t, solved.log_M, label = "log_M", lw = 3)
plot8 = plot(solved.t, solved.U, label = "U", lw = 3)
plot9 = plot(solved.t, solved._rat, label = "pi_rat", lw = 3)
plot(plot1, plot2, plot3, plot4, plot5, plot6, plot7, plot8, plot9, layout=(3,3), size
```

Out[27]:



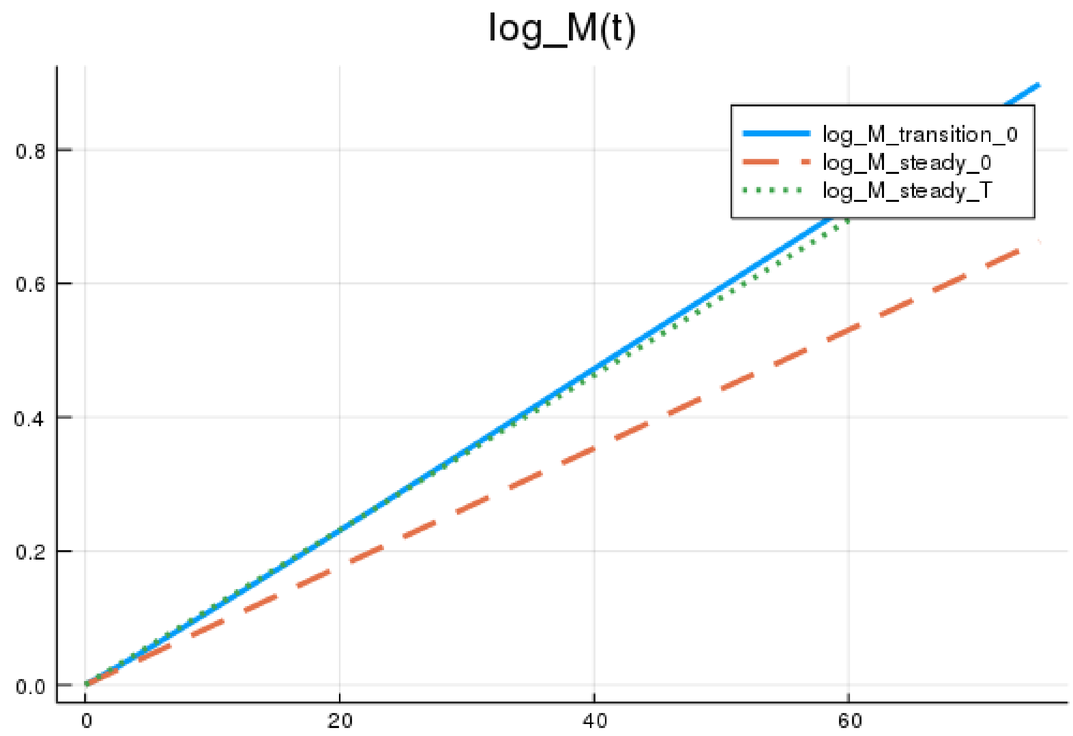
3 Welfare analysis

3.0.1 log_M(t)

In [28]:
```
# define log_M with steady state g
log_M_steady_0(t) = stationary_sol_0.g * t
log_M_steady_T(t) = stationary_sol.g * t

# generate the plot!
plot(solved.t,
    [solved.log_M, log_M_steady_0, log_M_steady_T],
    label = ["log_M_transition_0", "log_M_steady_0", "log_M_steady_T"] ,
    title = "log_M(t)", linestyle = :auto, lw = 3)
```

Out[28]:

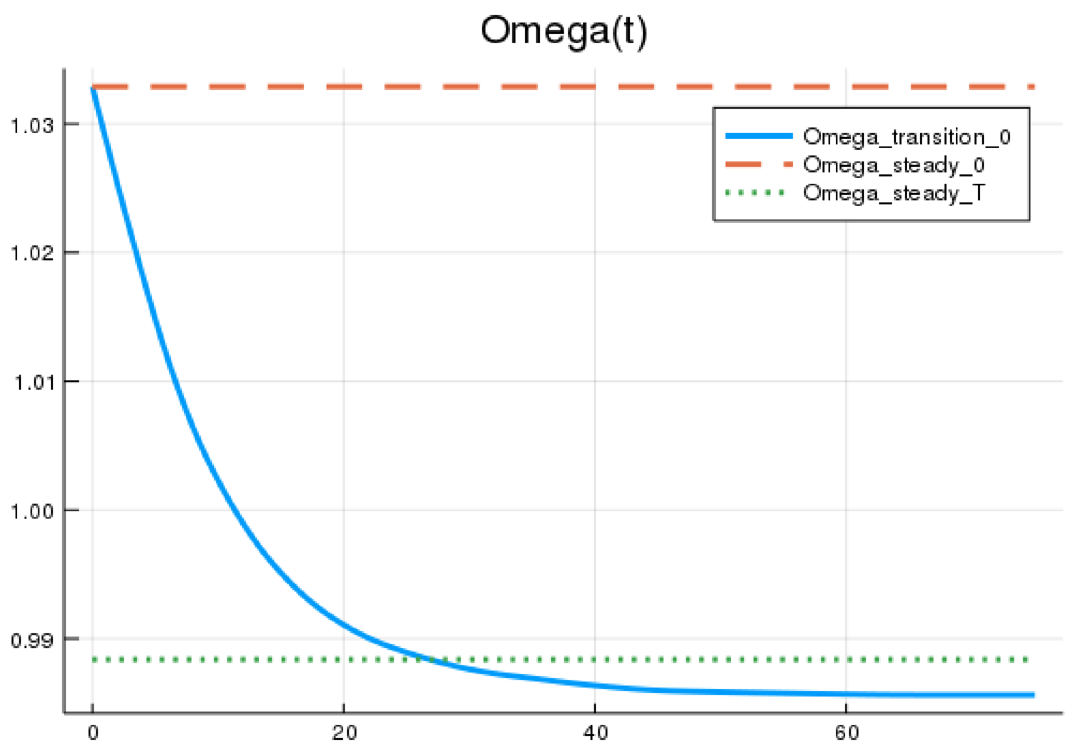


3.0.2 (t)

In [29]:
```
# define  function (constant)
_steady_0(t) = stationary_sol_0.
_steady_T(t) = stationary_sol.

# generate the plot!
plot(solved.t,
    [solved., _steady_0, _steady_T],
    label = ["Omega_transition_0", "Omega_steady_0", "Omega_steady_T"] ,
    title = "Omega(t)", linestyle = :auto, lw = 3)
```

Out[29]:

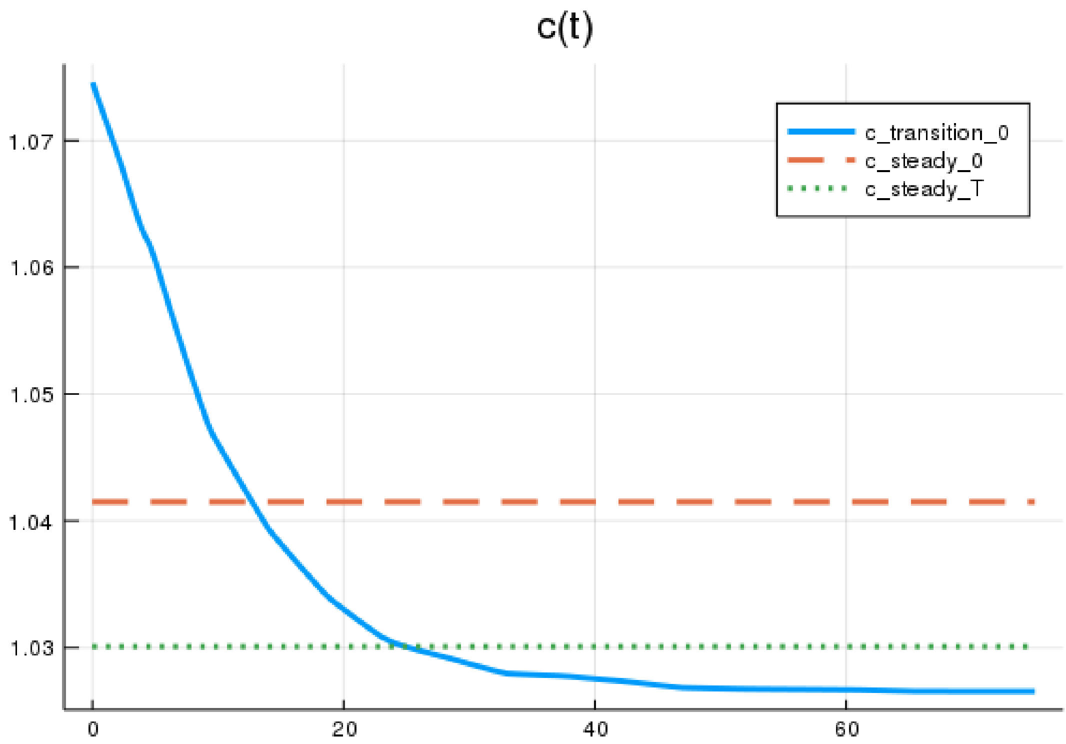


3.0.3 c(t)

```
In [30]: # define c function (constant)
         c_steady_0(t) = stationary_sol_0.c
         c_steady_T(t) = stationary_sol.c

         # generate the plot!
         plot(solved.t,
             [solved.c, c_steady_0, c_steady_T],
             label = ["c_transition_0", "c_steady_0", "c_steady_T"] ,
             title = "c(t)", linestyle = :auto, lw = 3)
```

Out[30]:

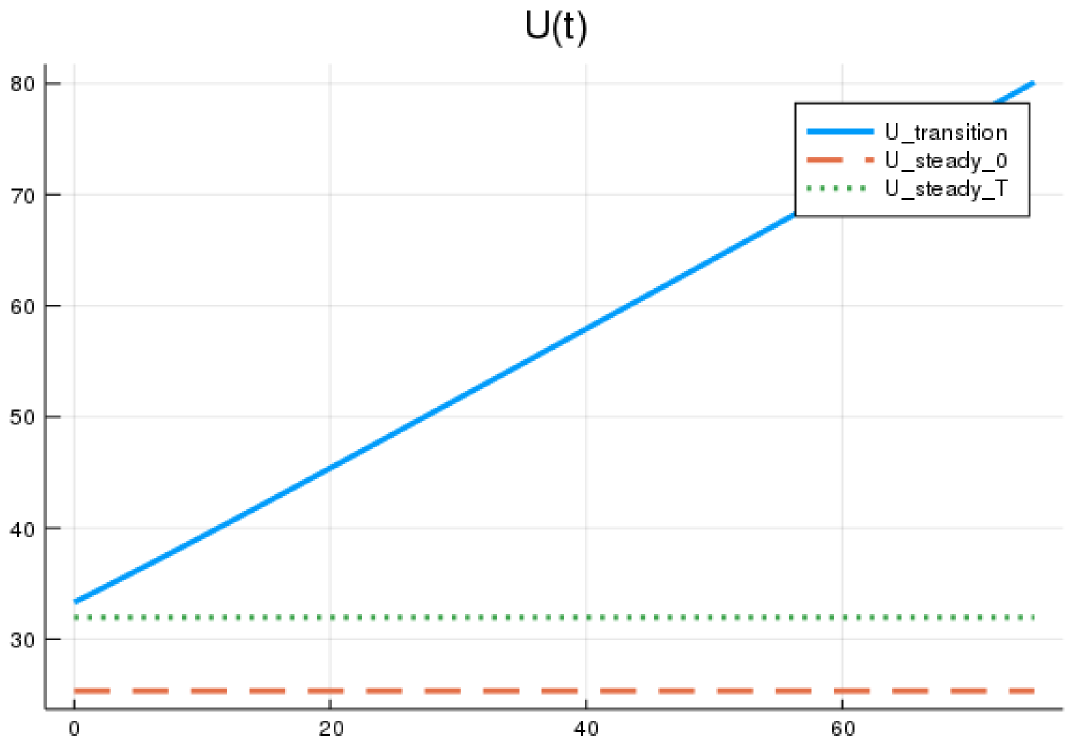


3.0.4 U(t)

```
In [31]: # generate the plot!
         U_steady_0(t) = stationary_sol_0.U_bar
         U_steady_T(t) = stationary_sol.U_bar

         plot(solved.t,
             [solved.U, U_steady_0, U_steady_T],
             label = ["U_transition", "U_steady_0", "U_steady_T"] ,
             title = "U(t)", linestyle = :auto, lw = 3)
```

Out[31]:



3.1 Consumption equivalent for search threshold (`M(0)`)

See computational appendix for details.

3.1.1 `M(0)` by two steady states (autarky and steady state at `T`)

In [32]:
```
T = solved.t[end]
consumption_equivalent(solved.U[end], stationary_sol_0.U_bar, parameters)
```

Out[32]: 2.9102176683529106

3.1.2 `M(0)` by autarky and transition from `t=0`

In [33]:
```
T = solved.t[end]
consumption_equivalent(solved.U[1], stationary_sol_0.U_bar, parameters)
```

Out[33]: 1.1680672085349637

But if we include the transition path, this falls a lot. Like down to 13 percent gain.

In [34]:
```
# solved
# Run the above to see the whole dataframe
```

In [35]:
```
using DataFrames

df_stationary = DataFrame(t = -1.00, g =stationary_sol_0.g, _ii = stationary_sol_0._ii
    _rat = stationary_sol_0._rat, L_tilde_a = stationary_sol_0.L_tilde_a, L_tilde_x =
```

In [36]:
```
CSV.write("stationary_results.csv", df_stationary)
```

Out[36]: "stationary_results.csv"

In [37]:
```
df_transition = DataFrame(t = solved.t, g =solved.g, _ii = solved._ii, c = solved.c, l
    _rat = solved._rat, L_tilde_a = solved.L_tilde_a, L_tilde_x = solved.L_tilde_x, L_
```

In [38]:
```
CSV.write("transition_results.csv", df_transition)
```

Out[38]: "transition_results.csv"

In [39]:
```
stationary_sol_0._rat
```

Out[39]: 1.7250561867295242

In [40]:
```
df_welfare = DataFrame(steady_state = lambda_ss, transition_path = lambda_tpath, grow
CSV.write("welfare_results.csv", df_welfare)
```

Out[40]: "welfare_results.csv"

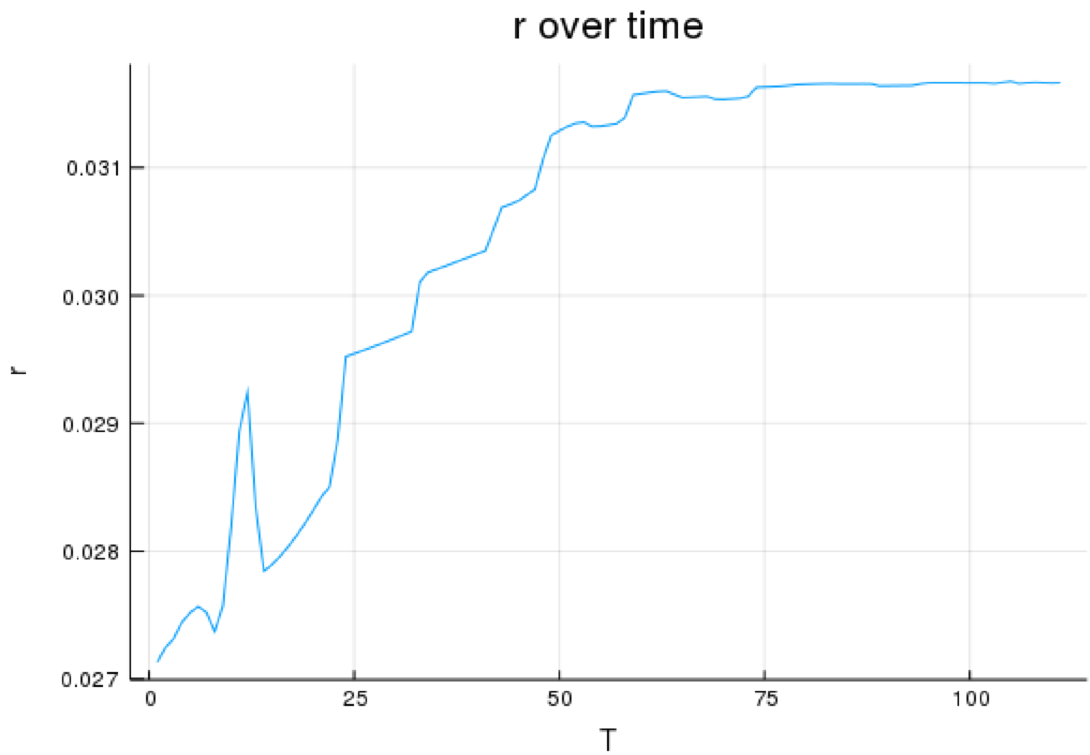
In [41]:
```
df_autarky = DataFrame(steady_state = lambda_ss_autarky, growth_rate = stationary_auta
CSV.write("autarky_welfare_results.csv", df_autarky)
```

Out[41]: "autarky_welfare_results.csv"

3.1.3 R and W

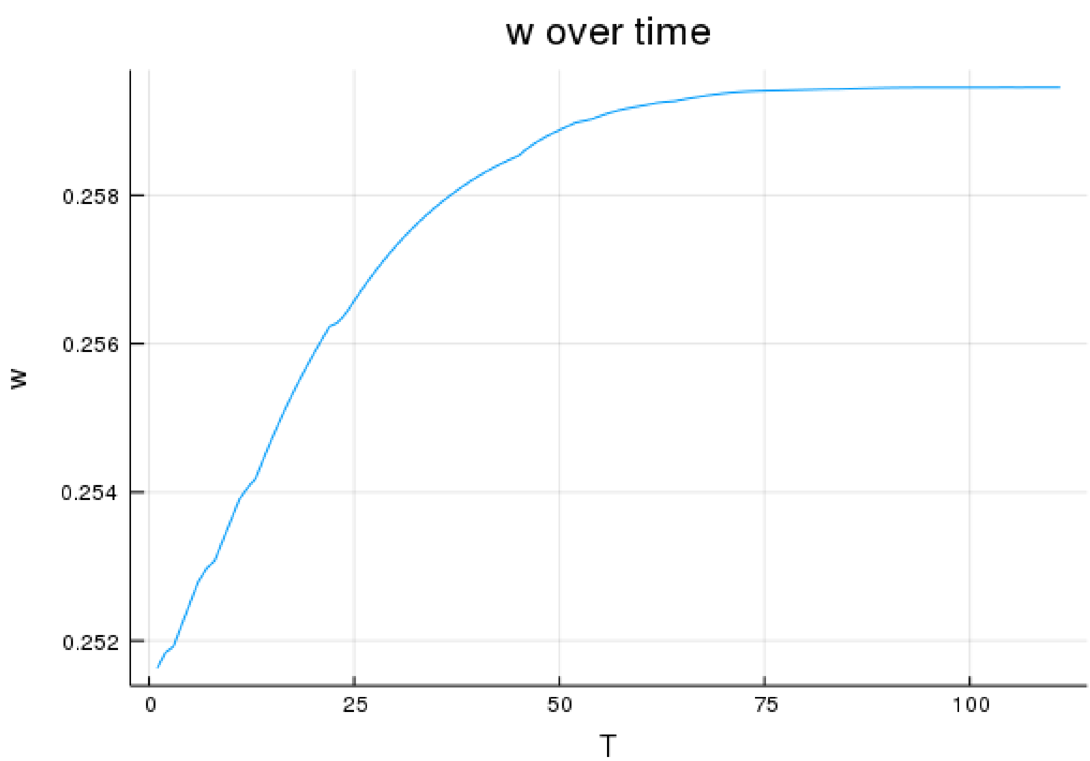
In [42]:
```
plot(solved.r, legend = false, title = "r over time", xlabel = "T", ylabel = "r")
```

Out[42]:



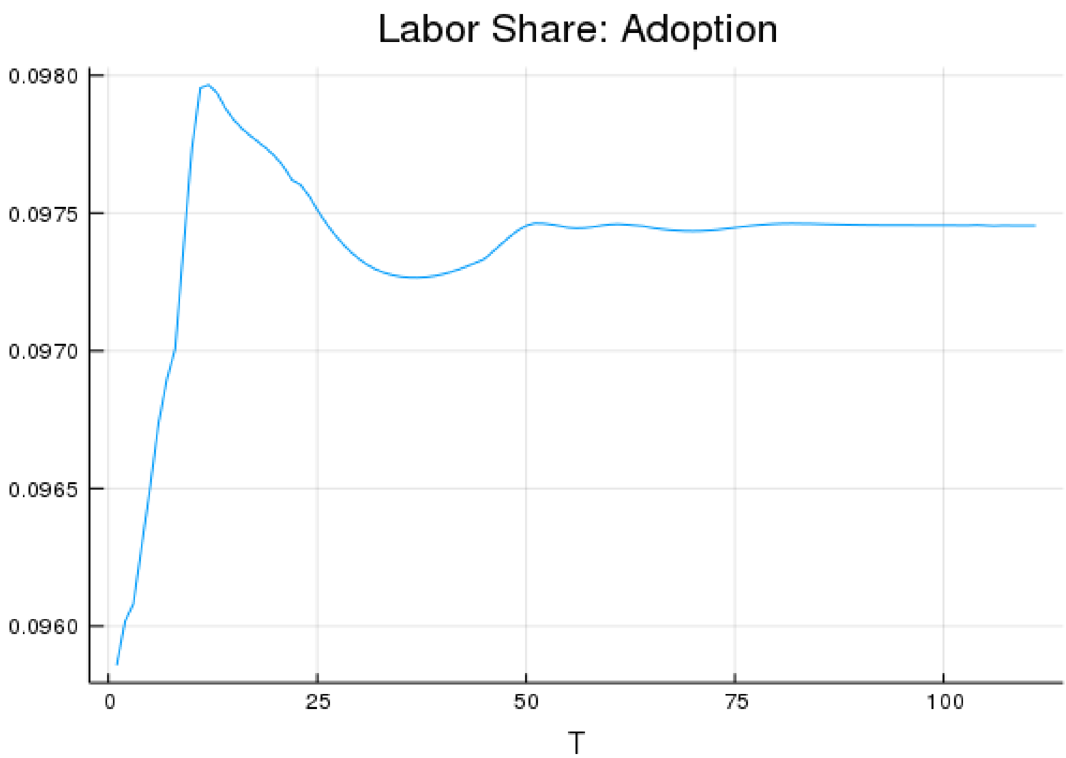

```
In [43]: plot(solved.w, legend = false, title = "w over time", xlabel = "T", ylabel = "w")
```

Out[43]:



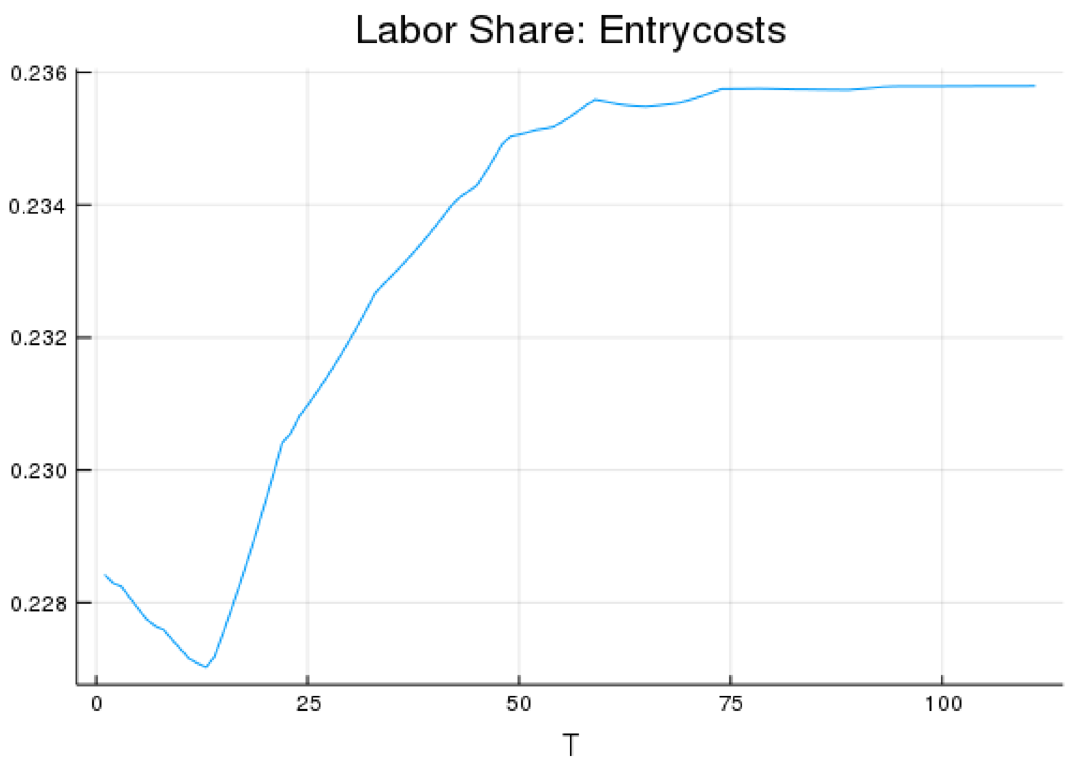
In [44]:
```
plot(solved.L_tilde_a, legend = false, xlabel = "T", title = "Labor Share: Adoption")
```

Out[44]:



In [45]:
```
plot(solved.L_tilde_E, legend = false, xlabel = "T", title = "Labor Share: Entrycosts"
```

Out[45]:




```
In [46]: plot(solved.L_tilde_x, legend = false, xlabel = "T", title = "Labor Share: Export")
```

Out[46]: