Untitled139

May 30, 2024

[21]:
```
import os
os.path.isfile(r'C:
 ↪\Users\USER\Documents\GitHub\GEModelToolsNotebooks\HANK-sticky-prices\HANKModel.
 ↪py')
```

[21]:
```
True
```

[22]:
```
import sys
print(sys.path)
```

```
['C:\\Users\\USER', 'C:\\Users\\USER\\anaconda3\\python311.zip',
'C:\\Users\\USER\\anaconda3\\DLLs', 'C:\\Users\\USER\\anaconda3\\Lib',
'C:\\Users\\USER\\anaconda3', '', 'C:\\Users\\USER\\anaconda3\\Lib\\site-
packages', 'C:\\Users\\USER\\anaconda3\\Lib\\site-packages\\win32',
'C:\\Users\\USER\\anaconda3\\Lib\\site-packages\\win32\\lib',
'C:\\Users\\USER\\anaconda3\\Lib\\site-packages\\Pythonwin',
'C:\\Users\\USER\\Documents\\GitHub\\GEModelToolsNotebooks\\HANK-sticky-prices',
'C:\\Users\\USER\\Documents\\GEModelTools-master']
```

[23]:
```
sys.path.append(r'C:
 ↪\Users\USER\Documents\GitHub\GEModelToolsNotebooks\HANK-sticky-prices')
```

[24]:
```
import sys
sys.path.append(r'C:\Users\USER\Documents\GEModelTools-master')
```

[25]:
```
%load_ext autoreload
%autoreload 2

import time
import pickle
import numpy as np
from scipy import optimize

import matplotlib.pyplot as plt
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
plt.rcParams.update({"axes.grid" : True, "grid.color": "black", "grid.alpha":"0.
 ↪25", "grid.linestyle": "--"})
plt.rcParams.update({'font.size': 14})
```

```
from HANKModel import HANKModelClass
```

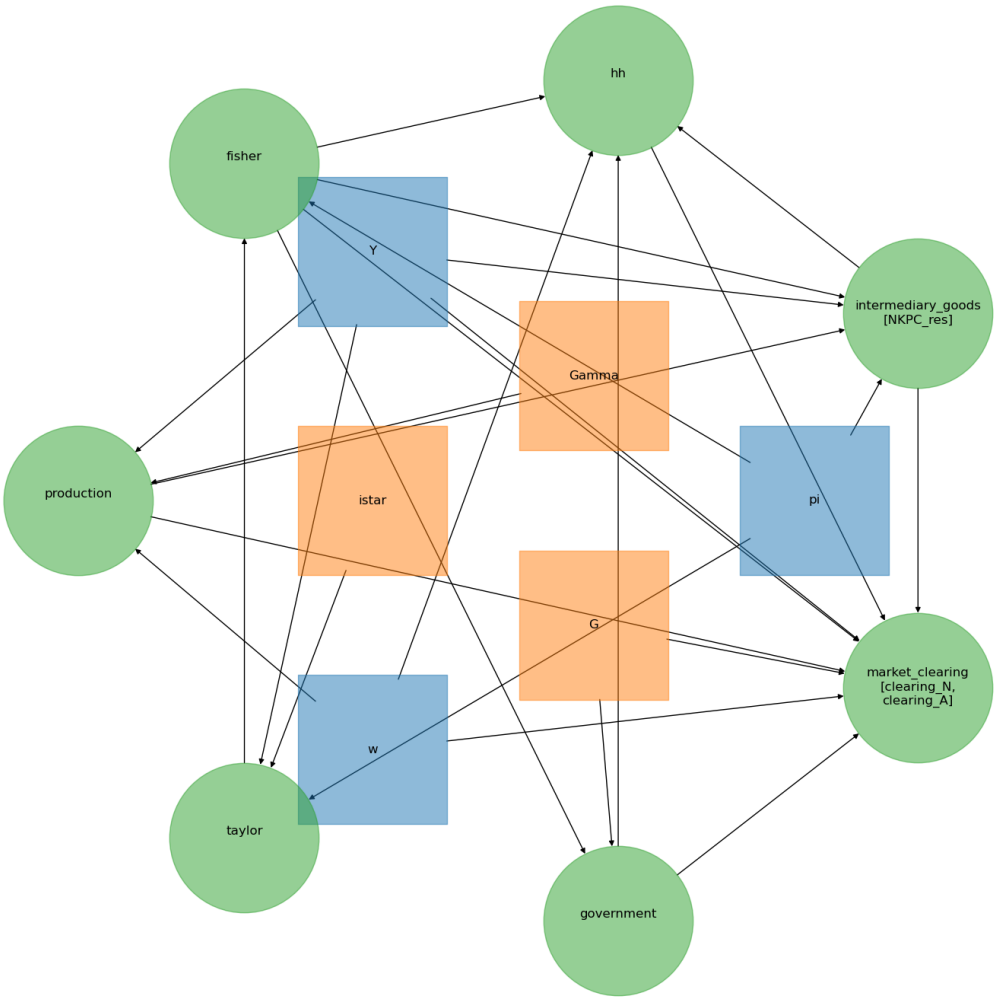
```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

[26]:
```
model = HANKModelClass(name='baseline')
```

[27]:
```
model.info()
```

```
settings:
 par.py_hh = False
 par.py_blocks = True
 par.full_z_trans = False
 par.warnings = True
 par.T = 1000

households:
 grids_hh: [a]
 pols_hh: [a]
 inputs_hh: [w,r,d,tau]
 inputs_hh_z: []
 outputs_hh: [a,c,ell,n]
 intertemps_hh: [vbeg_a]

aggregate:
 shocks: [Gamma,istar,G]
 unknowns: [Y,w,pi]
 targets: [NKPC_res,clearing_N,clearing_A]

blocks (inputs -> outputs):
 production: [Gamma,w,Y] -> [N,s]
 taylor: [istar,pi,Y] -> [i]
 fisher: [i,pi] -> [r]
 government: [G,r] -> [B,tau]
 intermediary_goods: [r,s,Y,pi] -> [NKPC_res,adjcost,d]
 hh: [d,r,tau,w] -> [A_hh,C_hh,ELL_hh,N_hh]
 market_clearing: [B,N,Y,G,adjcost,N_hh,A_hh,C_hh,r,w] ->
[A,clearing_N,clearing_A,clearing_Y]
```

[28]:
```
model.draw_DAG(figsize=(15,15),order=['shocks','unknowns','blocks'])
plt.savefig(r"C:\Users\USER\Documents\GitHub\New folder\DAG_image.png", dpi=600)

# Show the plot (optional, can be omitted if you only need the saved file)
plt.show()
```



[29]:
```
import pandas as pd

# Using raw string literals
df = pd.read_csv(r'C:\Users\USER\Documents\GitHub\hank sticky\qdatakorea.csv',␣
 ↪index_col=0)
print(df)
```

```
            w          r         d        tau
t
1990Q1   925.0  11.000000  15.02500  39.545944
1990Q2   925.0  11.250000  15.38325  38.899679
1990Q3   925.0  11.500000  15.74150  38.253414
1990Q4   925.0  11.750000  16.09975  37.607149
```

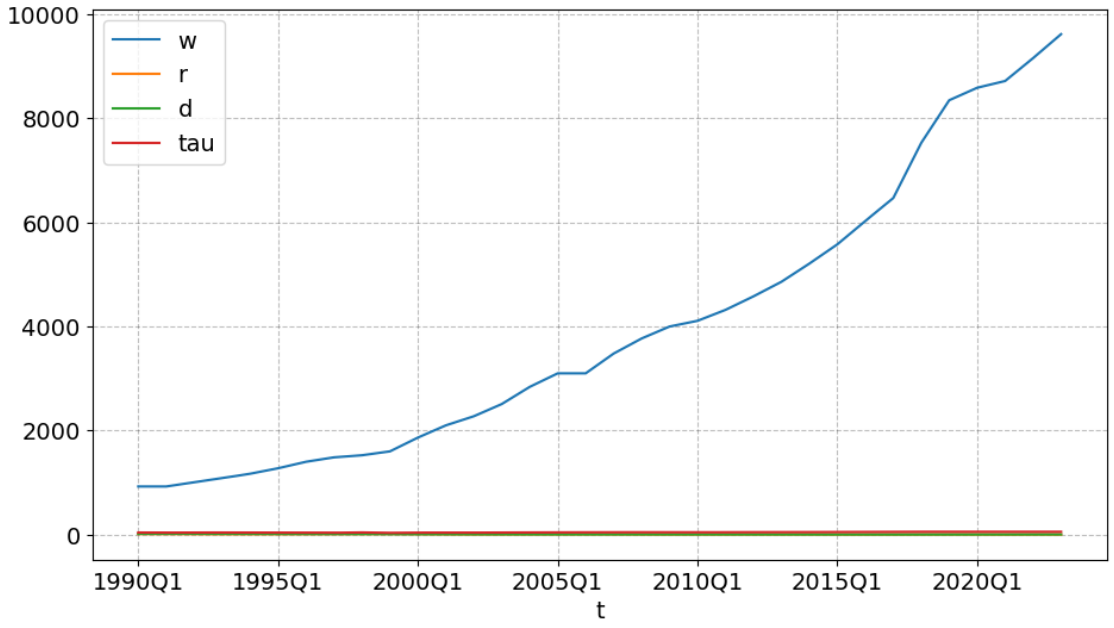
```
1991Q1   925.0  12.000000  16.45800  36.960884
…          …         …        …         …
2022Q1  9160.0   2.978980   3.47000  53.028748
2022Q2  9275.0   2.392347   3.55650  53.258468
2022Q3  9390.0   1.805715   3.64300  53.488189
2022Q4  9505.0   1.219082   3.72950  53.717909
2023Q1  9620.0   0.632449   3.81600  53.947629

[133 rows x 4 columns]
```

[30]:
```
# Convert 'w' column to numeric
df['w'] = pd.to_numeric(df['w'], errors='coerce')
```

[31]:
```
# Convert 'w' column to numeric
df['w'] = pd.to_numeric(df['w'], errors='coerce')
```

[32]:
```
# Set the DPI to 2000
plt.figure(dpi=600)
df.plot(figsize=(11,6));
# Save the figure with the specified name and DPI
plt.savefig(r"C:\Users\USER\Documents\GitHub\hank sticky\df.png")

plt.show()
```

```
<Figure size 3840x2880 with 0 Axes>
```



[33]:
```
par = model.par
ss = model.ss
path = model.path
sim = model.sim
```

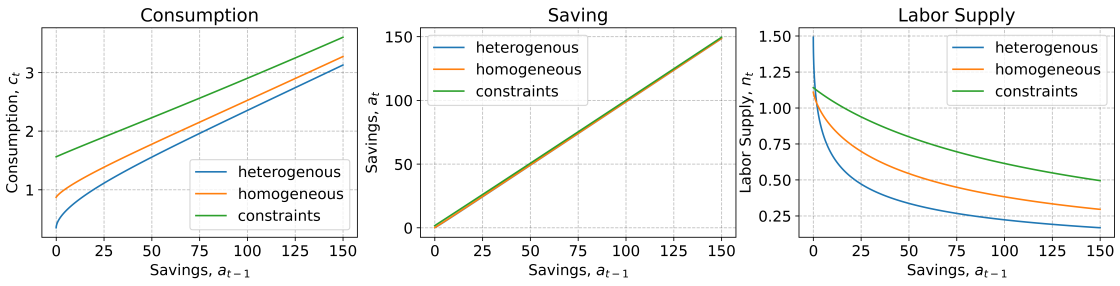
[34]:
```
import pandas as pd
ss.w = df['w']
ss.r = df['r']
ss.d = df['d']
ss.tau = df['tau']
```

[35]:
```
model.find_ss(do_print=True)
```

```
steady state found in 11.2 secs
 beta    =   0.9773
 varphi  =   0.7889

Discrepancy in A =   0.00000000
Discrepancy in N =   0.00000000
Discrepancy in Y =   0.00000000
```

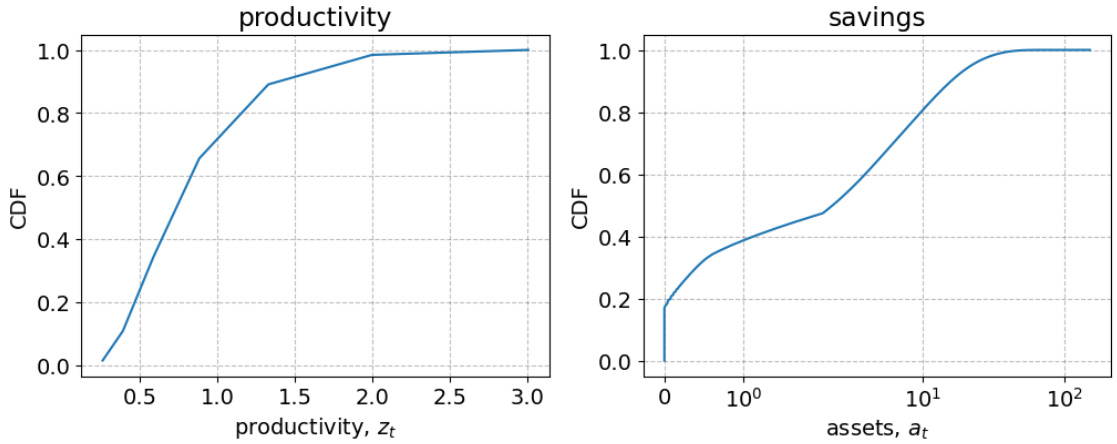
[37]:
```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(18, 4), dpi=600)
a_max = 500

# a. consumption
I = par.a_grid < a_max

ax = fig.add_subplot(1, 3, 1)
ax.set_title('Consumption')

curve_names = ['heterogenous', 'homogeneous', 'constraints']

for i, i_z in enumerate([0, par.Nz // 2, par.Nz - 1]):
    ax.plot(par.a_grid[I], ss.c[0, i_z, I], label=curve_names[i])

ax.legend(frameon=True)
ax.set_xlabel('Savings, $a_{t-1}$')
ax.set_ylabel('Consumption, $c_t$')

# b. saving
ax = fig.add_subplot(1, 3, 2)
ax.set_title('Saving')

for i, i_z in enumerate([0, par.Nz // 2, par.Nz - 1]):
    ax.plot(par.a_grid[I], ss.a[0, i_z, I], label=curve_names[i])
```

```
ax.legend(frameon=True)
ax.set_xlabel('Savings, $a_{t-1}$')
ax.set_ylabel('Savings, $a_{t}$')

# c. labor supply
ax = fig.add_subplot(1, 3, 3)
ax.set_title('Labor Supply')

for i, i_z in enumerate([0, par.Nz // 2, par.Nz - 1]):
    ax.plot(par.a_grid[I], ss.ell[0, i_z, I], label=curve_names[i])

ax.legend(frameon=True)
ax.set_xlabel('Savings, $a_{t-1}$')
ax.set_ylabel('Labor Supply, $n_{t}$')

plt.subplots_adjust(bottom=0.2)  # Adjust the bottom margin

plt.savefig(r"C:\Users\USER\Documents\GitHub\New folder\consumption.png",␣
 ↪dpi=600)
plt.show()
```



[38]:
```
fig = plt.figure(figsize=(12,4),dpi=100)

# a. income
ax = fig.add_subplot(1,2,1)
ax.set_title('productivity')

y = np.cumsum(np.sum(ss.D[0],axis=1))
ax.plot(par.z_grid,y/y[-1])

ax.set_xlabel('productivity, $z_{t}$')
ax.set_ylabel('CDF')

# b. assets
ax = fig.add_subplot(1,2,2)
ax.set_title('savings')
```

```
y = np.insert(np.cumsum(np.sum(ss.D[0],axis=0)),0,0.0)
ax.plot(np.insert(par.a_grid,0,par.a_grid[0]),y/y[-1])

ax.set_xlabel('assets, $a_{t}$')
ax.set_ylabel('CDF')
ax.set_xscale('symlog')
plt.savefig(r"C:\Users\USER\Documents\GitHub\hank sticky\productivity.png",␣
 ↪dpi=500)
plt.show()
```

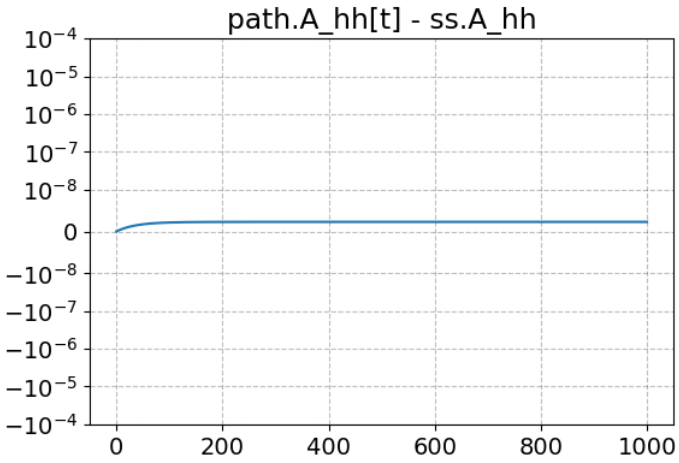


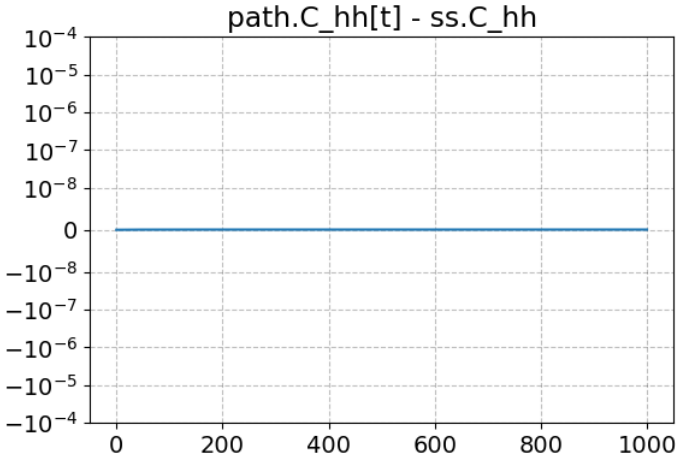
[39]:
```
print(f'{model.ss.A_hh = :.2f}')
print(f'{model.ss.ELL_hh = :.2f}')
print(f'{model.ss.C_hh = :.2f}')
print(f'{model.ss.N_hh = :.2f}')
```

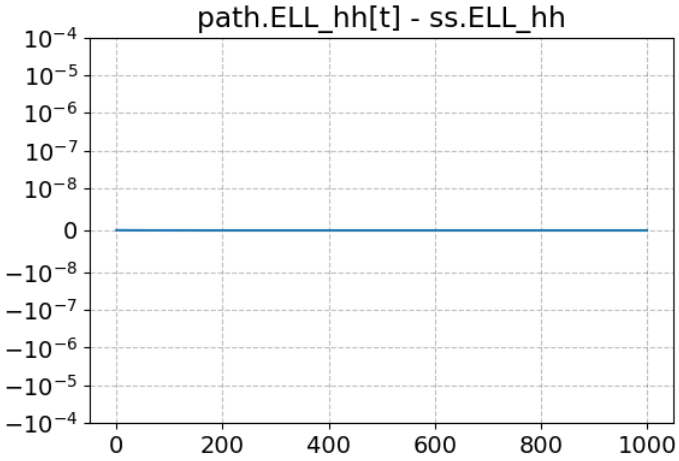
```
model.ss.A_hh = 5.60
model.ss.ELL_hh = 1.04
model.ss.C_hh = 1.00
model.ss.N_hh = 1.00
```

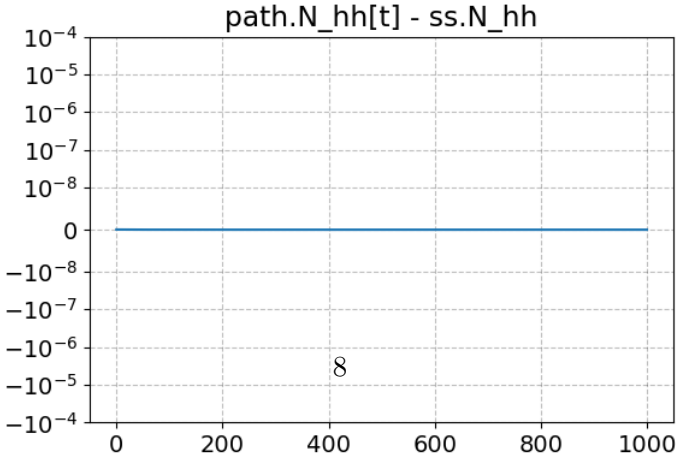
[40]:
```
model.test_hh_path()
```

```
note: inputs = steady state value -> expected: constant value (straigt line) in
roughly -10^-5 to 10^5

household problem solved along transition path in 0.9 secs
household problem simulated along transition in 1.1 secs
```


path.A_hh[t] - ss.A_hh
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
0
-10^{-8}
-10^{-7}
-10^{-6}
-10^{-5}
-10^{-4}
0
200
400
600
800
1000


path.C_hh[t] - ss.C_hh
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
0
-10^{-8}
-10^{-7}
-10^{-6}
-10^{-5}
-10^{-4}
0
200
400
600
800
1000


path.ELL_hh[t] - ss.ELL_hh
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
0
-10^{-8}
-10^{-7}
-10^{-6}
-10^{-5}
-10^{-4}
0
200
400
600
800
1000


path.N_hh[t] - ss.N_hh
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
0
-10^{-8}
-10^{-7}
-10^{-6}
-10^{-5}
-10^{-4}
0
200
400
600
800
1000

[41]:
```
model.find_ss(do_print=True)
```

```
steady state found in 3.4 secs
 beta   =   0.9773
 varphi =   0.7889

Discrepancy in A =  -0.00000000
Discrepancy in N =   0.00000000
Discrepancy in Y =   0.00000000
```

[42]:
```
try:
      model.test_ss()
except Exception as e:
      print('you need to update GEModelTools to call this function (optional)')
```

```
Gamma          :       1.0000
w              :       0.8333
Y              :       1.0000
N              :       1.0000
s              :       0.8333
istar          :       0.0100
pi             :       0.0000
i              :       0.0100
r              :       0.0100
G              :       0.0000
B              :       5.6000
tau            :       0.0560
NKPC_res       : nan
adjcost        :       0.0000
d              :       0.1667
A_hh           :       5.6000
C_hh           :       1.0000
ELL_hh         :       1.0359
N_hh           :       1.0000
A              :       5.6000
clearing_N     :       0.0000
clearing_A     :      -0.0000
clearing_Y     :       0.0000

C:\Users\USER\Documents\GEModelTools-master\GEModelTools\tests.py:19:
UserWarning: warning: NKPC_res contains nan
  if do_warnings: warnings.warn(f'warning: {varname} contains nan')
```

[43]:
```
model.test_path(in_place=True)
```

```
shocks: Gamma istar G
unknowns: Y w pi
```

```
look at max(abs(path.VARNAME[:]-ss.VARNAME)):

 blocks.production
  N                0.0e+00
  s                0.0e+00
 blocks.taylor
  i                0.0e+00
 blocks.fisher
  r                8.7e-18
 blocks.government
  B                0.0e+00
  tau              4.9e-17
 blocks.intermediary_goods
  NKPC_res         0.0e+00 [target]
  adjcost          0.0e+00
  d                0.0e+00
 hh
  A_hh             2.3e-09
  C_hh             5.2e-11
  ELL_hh           5.1e-11
  N_hh             4.2e-11
 blocks.market_clearing
  A                0.0e+00
  clearing_N       4.2e-11 [target]
  clearing_A       2.3e-09 [target]
  clearing_Y       5.2e-11
```

[44]:
```
model.compute_jacs(do_print=True)
```

```
household Jacobians:
one step deviation from steady state calculated in 0.0 secs
curly_Y and curly_D calculated for d               in 1.1 secs
curly_Y and curly_D calculated for r               in 1.1 secs
curly_Y and curly_D calculated for tau             in 1.1 secs
curly_Y and curly_D calculated for w               in 1.1 secs
curly_E calculated in 1.9 secs
builiding blocks combined in 0.7 secs
household Jacobian computed in 7.1 secs

full Jacobians:
full Jacobian to unknowns computed in 3.0 secs [in evaluate_blocks(): 1.3 secs]
full Jacobian to shocks computed in 2.8 secs [in evaluate_blocks(): 1.2 secs]
```

[45]:
```
model.find_transition_path(shocks=[],do_print=True)
```

```
finding the transition path:
 it =   0 -> max. abs. error = 2.28e-09
   0.00e+00 in NKPC_res
```

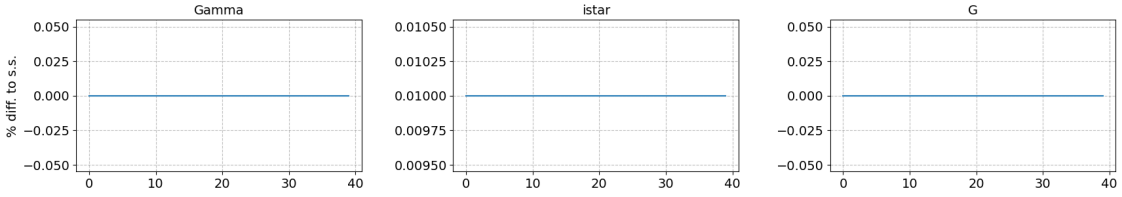
```
  4.20e-11 in clearing_N
  2.28e-09 in clearing_A
it =   1 -> max. abs. error = 6.16e-13
  2.25e-15 in NKPC_res
  2.82e-14 in clearing_N
  6.16e-13 in clearing_A

transition path found in 3.0 secs
```

[46]:
```
import matplotlib.pyplot as plt

# Switch to the default interactive backend for Jupyter Notebook
plt.switch_backend('module://ipykernel.pylab.backend_inline')

# Now try to plot and display the figure again
```

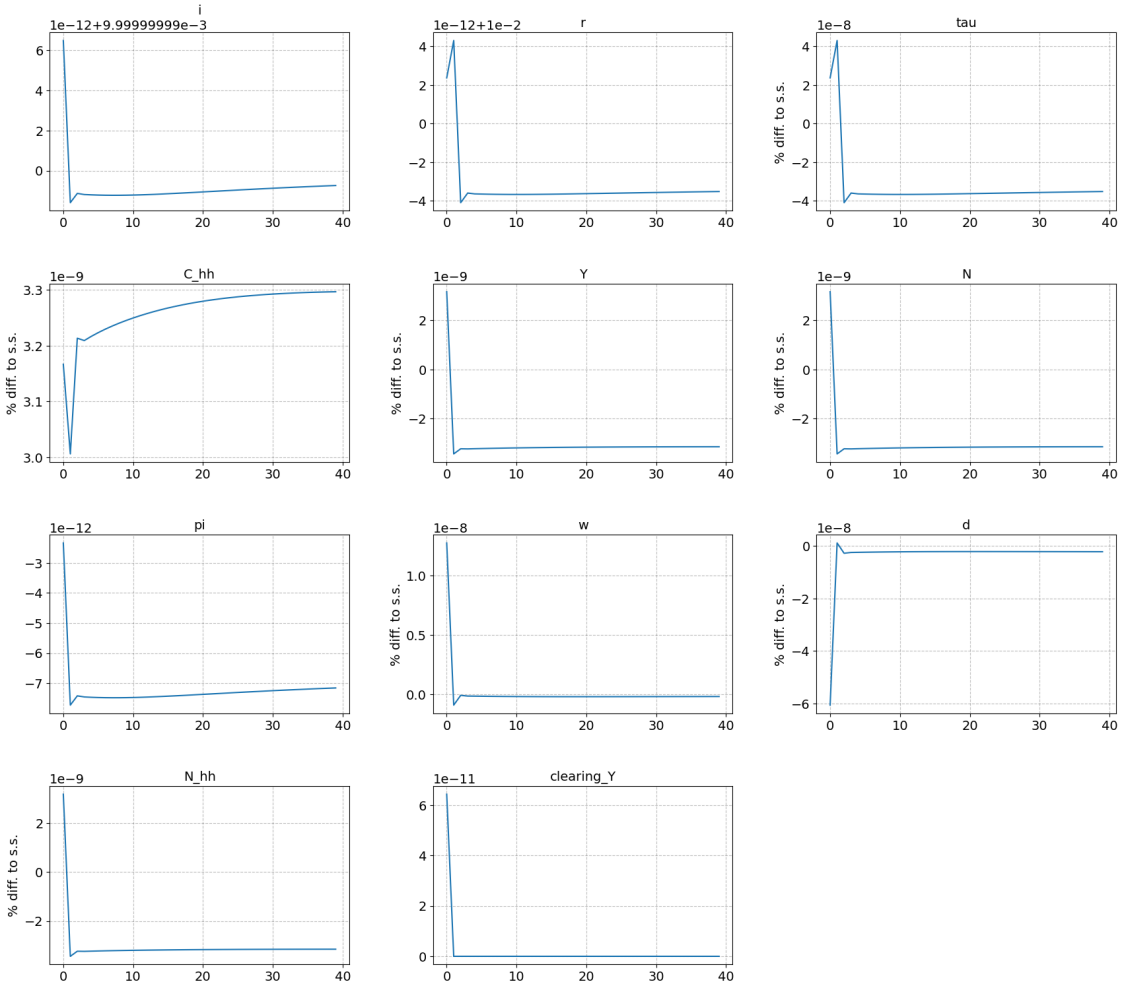
[47]:
```
import matplotlib.pyplot as plt

# Set the backend for interactive plotting
plt.switch_backend('module://ipykernel.pylab.backend_inline')
```

[68]:
```
paths = ['i','r','tau','C_hh','Y','N','pi','w','d','N_hh','clearing_Y']
lvl_value = ['i','pi','r','istar','G','clearing_Y']
model.show_IRFs(paths,lvl_value=lvl_value,T_max=40,ncols=3)
```

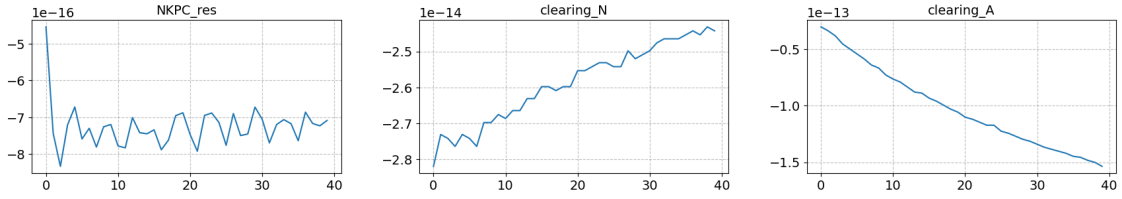
```
### shocks ###
```




```
### varnames ###
```



tagets

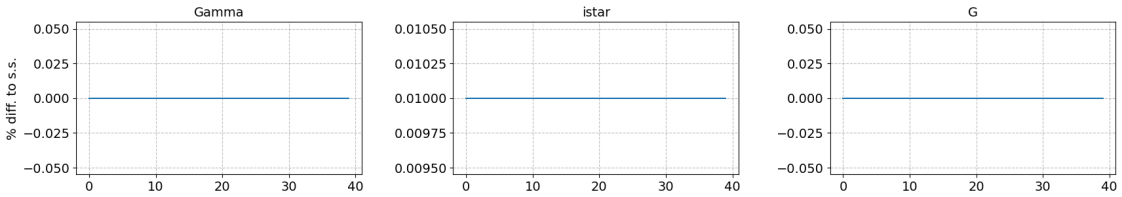


[72]:
```
# Set the DPI to 600
plt.figure(figsize=(18, 12), dpi=600)
paths = ['Y','r','C_hh']
```

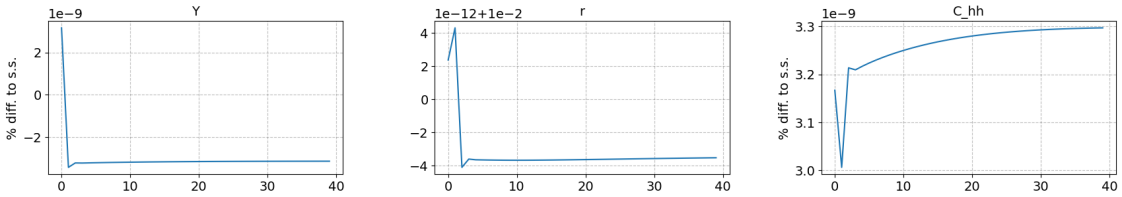
```
lvl_value = ['i','pi','r','istar','G','clearing_Y']
model.show_IRFs(paths,lvl_value=lvl_value,T_max=40,ncols=3)
```

```
### shocks ###
```

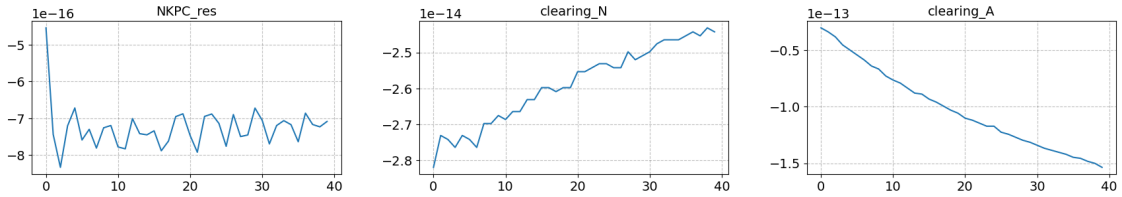
```
<Figure size 10800x7200 with 0 Axes>
```




```
### varnames ###
```



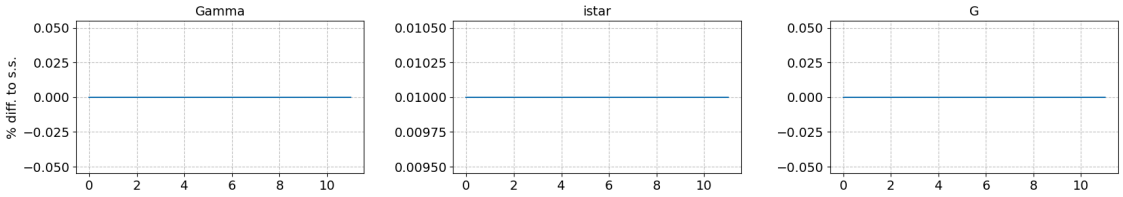

```
### tagets ###
```



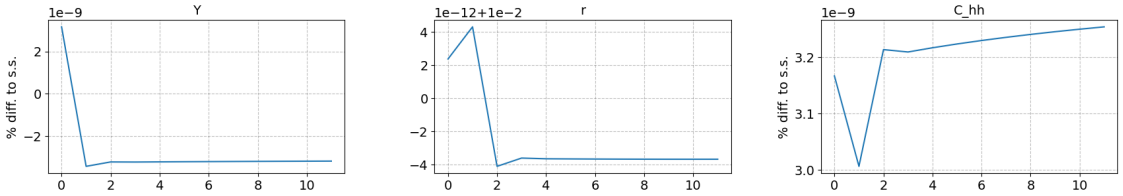
[55]:
```
# Set the DPI to 600
plt.figure(figsize=(18, 12), dpi=600)
paths = ['Y','r','C_hh']
lvl_value = ['i','pi','r','istar','G','clearing_Y']
model.show_IRFs(paths,lvl_value=lvl_value,T_max=12,ncols=3)
```

```
### shocks ###
```

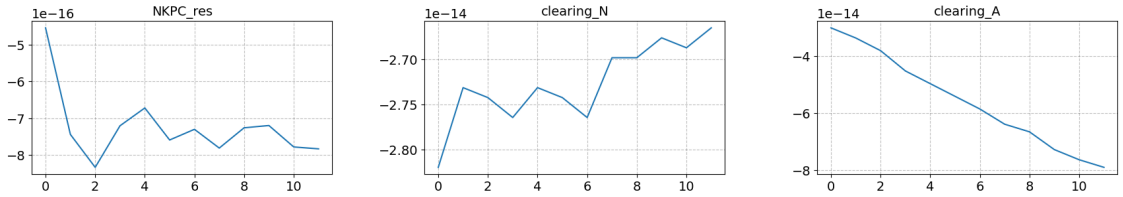
```
<Figure size 10800x7200 with 0 Axes>
```



varnames



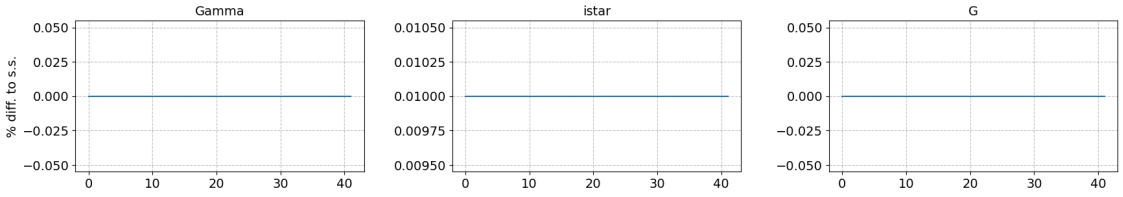
tagets



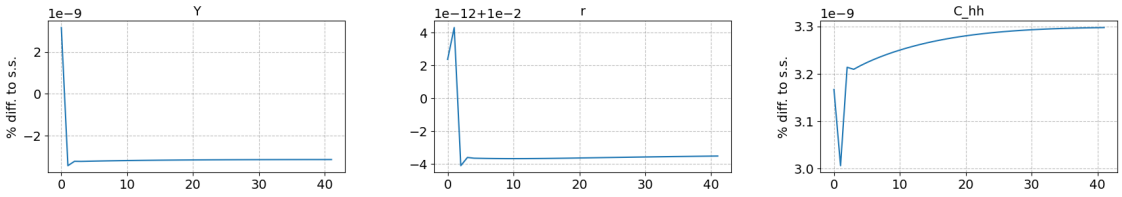
[73]:
```
# Set the DPI to 600
plt.figure(figsize=(18, 12), dpi=600)
paths = ['Y','r','C_hh']
lvl_value = ['i','pi','r','istar','G','clearing_Y']
model.show_IRFs(paths,lvl_value=lvl_value,T_max=42,ncols=4)
```

shocks

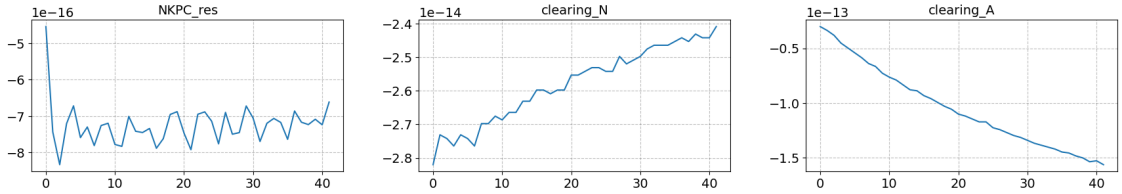
<Figure size 10800x7200 with 0 Axes>



varnames



tagets

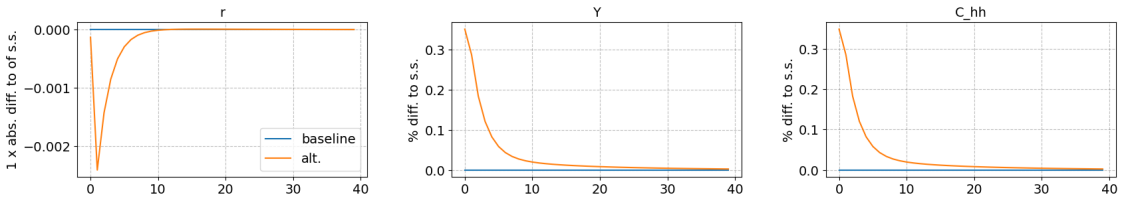


[93]:
```
# Set the DPI to 500
plt.figure(figsize=(18, 12), dpi=2000)
paths = ['Y','r','C_hh']
lvl_value = ['i','pi','r','istar','G','clearing_Y']
model.show_IRFs(paths,lvl_value=lvl_value,T_max=22,ncols=3)
# Save the current plot as PNG or JPEG image in landscape orientation
    image_path = rf'C:\Users\USER\Documents\GitHub\New folder\20q_plot_{path}.
 ↪{image_format}'
    plt.savefig(image_path, dpi=600, orientation='landscape',␣
 ↪bbox_inches='tight')
    plt.close()  # Close the current figure to free up resources
```

```
  Cell In[93], line 7
    image_path = rf'C:\Users\USER\Documents\GitHub\New folder\20q_plot_{path}.
 ↪{image_format}'
    ^
IndentationError: unexpected indent
```

[94]:
```
# Set the DPI to 2000
plt.figure(dpi=2000)
model_ = model.copy()
model_.par.kappa = 0.0025
model_.compute_jacs(skip_hh=True,skip_shocks=True)
model_.find_transition_path(shocks=['istar'])
paths = ['r','Y','C_hh']
model.compare_IRFs([model,model_],['baseline','alt.'],
                   ␣
 ↪paths,lvl_value,do_shocks=False,do_targets=False,ncols=3,T_max=40)
# Save the figure directly using savefig without displaying
plt.savefig(r"C:\Users\USER\Documents\GitHub\hank sticky\kappa40.png", dpi=500)

# Show the plot
plt.show()
```

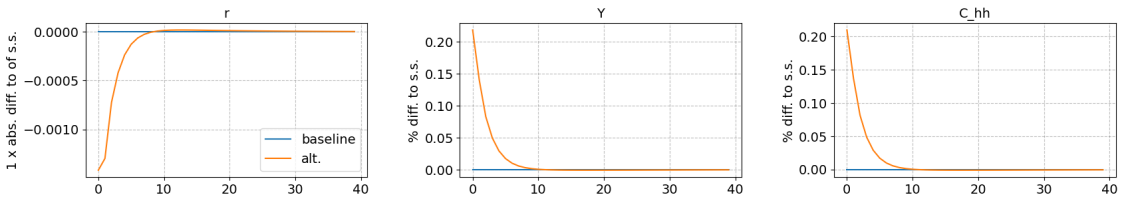
```
<Figure size 12800x9600 with 0 Axes>
```




```
<Figure size 640x480 with 0 Axes>
```

[33]:
```
# Set the DPI to 2000
plt.figure(dpi=2000)
model_ = model.copy()
model_.par.kappa = 0.07
model_.compute_jacs(skip_hh=True,skip_shocks=True)
model_.find_transition_path(shocks=['istar'])
paths = ['r','Y','C_hh']
model.compare_IRFs([model,model_],['baseline','alt.'],
                   ␣
 ↪paths,lvl_value,do_shocks=False,do_targets=False,ncols=3,T_max=40)
```

```
<Figure size 12800x9600 with 0 Axes>
```

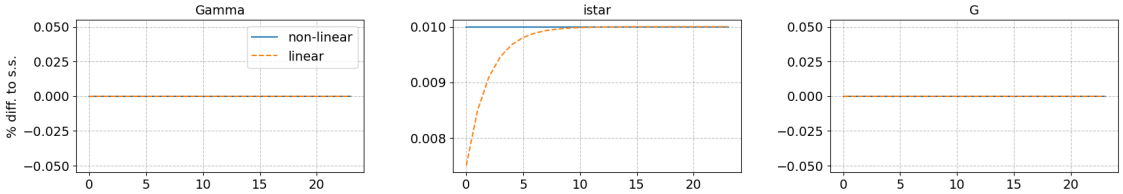


[35]:
```
model.find_IRFs(shocks=['istar'],do_print=True)
```

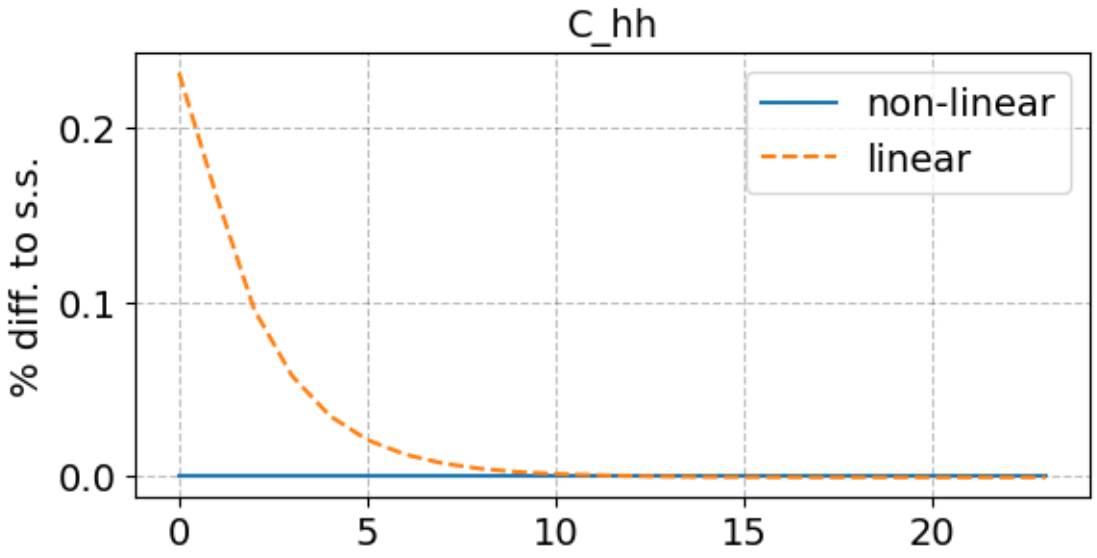
```
linear transition path found in 0.7 secs [finding solution matrix: 0.7 secs]
```

[37]:
```
# Set the DPI to 2000
plt.figure(dpi=2000)
paths = ['C_hh']
model.show_IRFs(paths,lvl_value=lvl_value,T_max=24,ncols=3,do_linear=True)
```

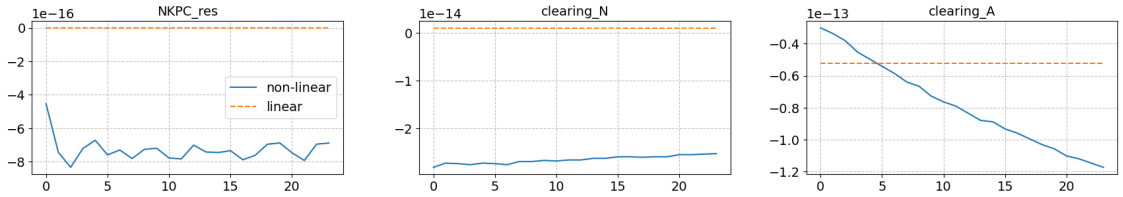
```
### shocks ###

<Figure size 12800x9600 with 0 Axes>
```




```
### varnames ###
```



```
### tagets ###
```

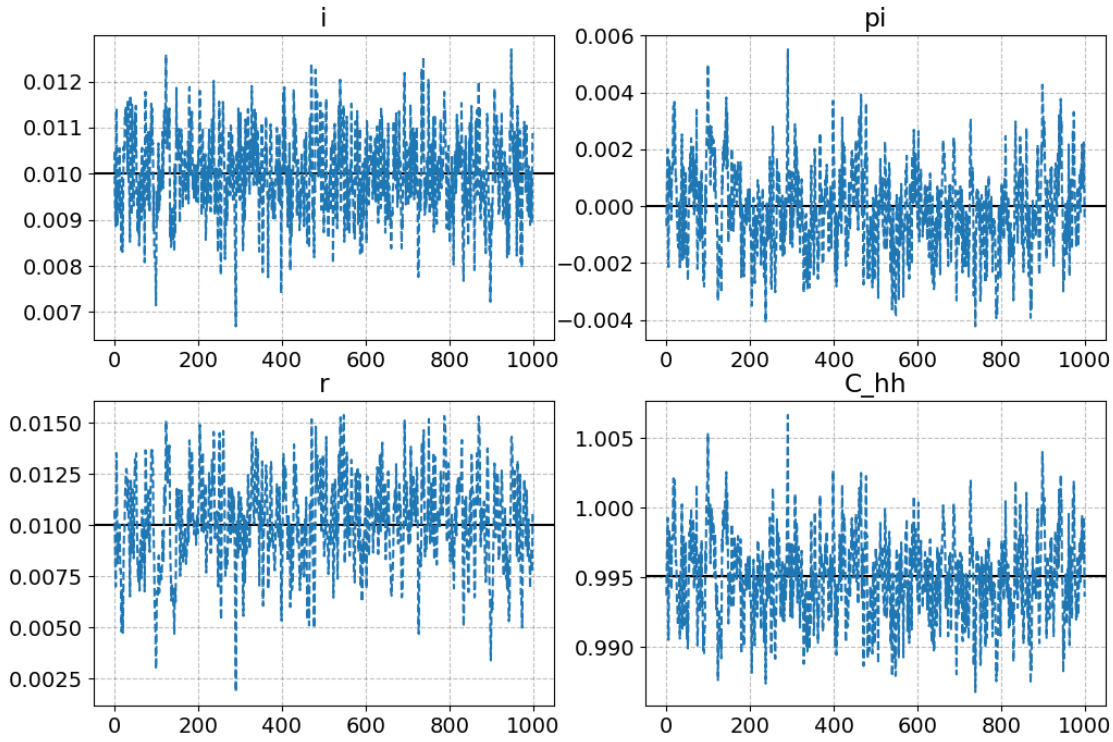


[38]:
```
np.random.seed(1917)
model.simulate(do_print=True)
```

```
simulation prepared in 50.2 secs [solution matrix: 0.7 secs, households: 49.4
secs]
aggregates simulated in 0.6 secs
household policies simulated in 14.1 secs
distribution simulated in 0.2 secs
aggregates calculated from distribution in 0.0 secs
```

[39]:
```
def model_sim():

    fig = plt.figure(figsize=(12,8),dpi=100)

    ax = fig.add_subplot(2,2,1)
    ax.set_title('i')
    ax.plot(ss.i+sim.di,ls='--')
    ax.axhline(ss.i,color='black',zorder=0)

    ax = fig.add_subplot(2,2,2)
    ax.set_title('pi')
    ax.plot(ss.pi+sim.dpi,ls='--',)
    ax.axhline(ss.pi,color='black',zorder=0)

    ax = fig.add_subplot(2,2,3)
    ax.set_title('r')
    ax.plot(ss.r+sim.dr,ls='--',)
    ax.axhline(ss.r,color='black',zorder=0)

    ax = fig.add_subplot(2,2,4)
    ax.set_title('C_hh')
    ax.plot(ss.C_hh+sim.dC_hh,ls='--',)
    ax.axhline(ss.C_hh,color='black',zorder=0)
```

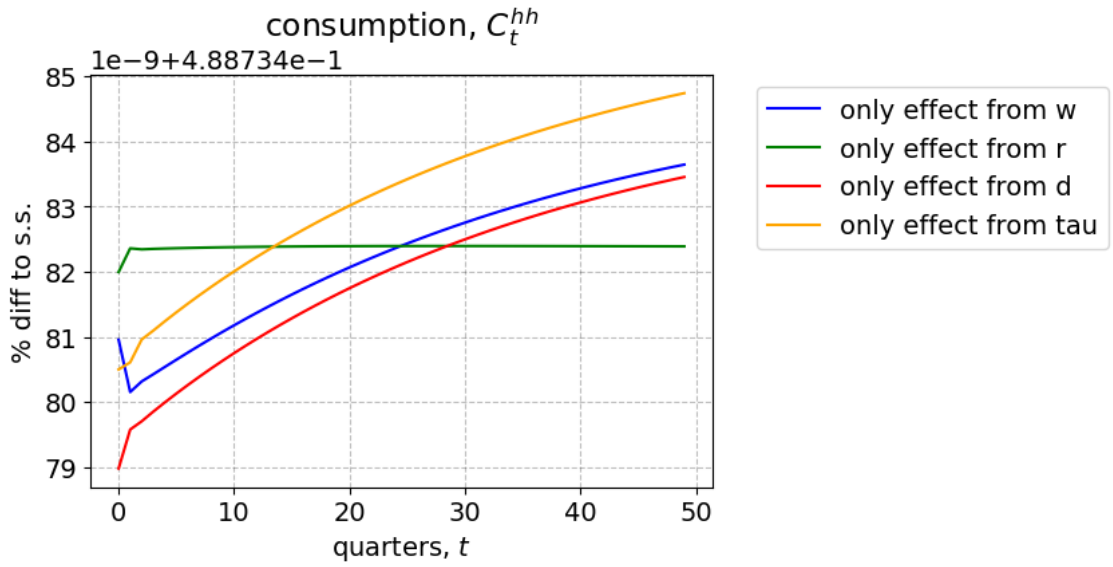
[109]:
```
model_sim()
# Save the figure directly using savefig without displaying
plt.savefig(r"C:\Users\USER\Documents\GitHub\hank sticky\simu", dpi=500)

# Show the plot
plt.show()
```

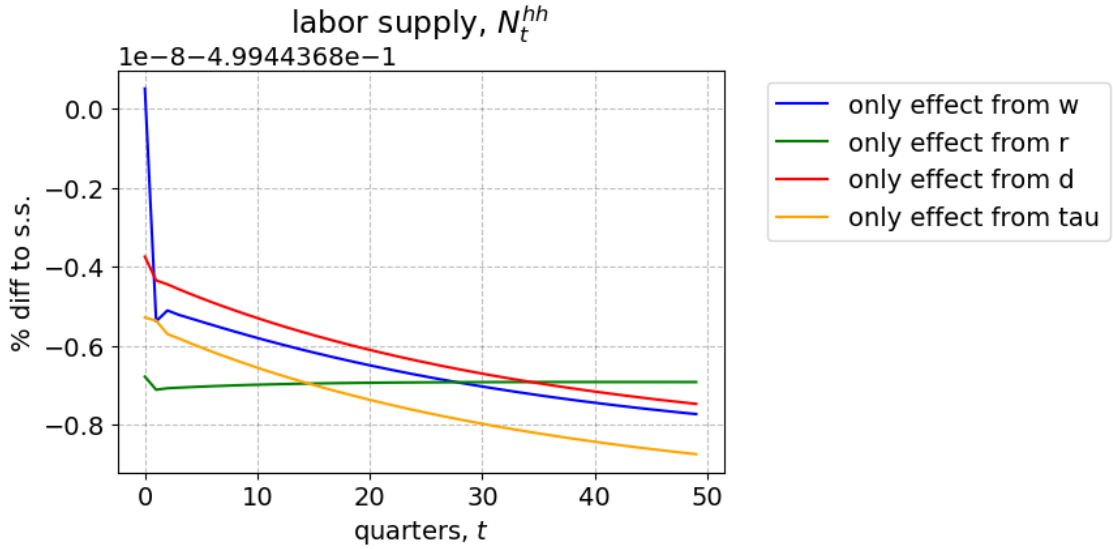


[118]:
```
T_max = 50

fig_C = plt.figure(figsize=(6,4),dpi=100)
ax_C = fig_C.add_subplot(1,1,1)
ax_C.set_title('consumption, $C_t^{hh}$')

fig_N = plt.figure(figsize=(6,4),dpi=100)
ax_N = fig_N.add_subplot(1,1,1)
ax_N.set_title('labor supply, $N_t^{hh}$')

i_color = 0
for use_inputs in [[x] for x in model.inputs_hh]:

    # a. compute
    print(use_inputs)
    path_alt = model.decompose_hh_path(do_print=True,use_inputs=use_inputs)
```

```
    print('')

    # b. plot
    if use_inputs is None:
        label = 'no inputs'
        ls = '--'
        color = 'black'
    elif use_inputs == 'all':
        label = 'all inputs'
        ls = '-'
        color = 'black'
    else:
        label = f'only effect from {use_inputs[0]}'
        ls = '-'
        color = colors[i_color]
        i_color += 1

    ax_C.plot((path_alt.C_hh[:T_max]/ss.
 ↪C_hh-1)*100,ls=ls,color=color,label=label);
    ax_N.plot((path_alt.N_hh[:T_max]/ss.
 ↪N_hh-1)*100,ls=ls,color=color,label=label);

for ax in [ax_C,ax_N]:
    ax.set_ylabel('% diff to s.s.')
    ax.set_xlabel('quarters, $t$')
    lgd = ax.legend(frameon=True,ncol=1,bbox_to_anchor=(1.05,1), loc='upper␣
 ↪left',)

plt.show()
```

```
['w']
household problem solved along transition path in 0.8 secs
household problem simulated along transition in 0.1 secs

['r']
household problem solved along transition path in 0.8 secs
household problem simulated along transition in 0.3 secs

['d']
household problem solved along transition path in 0.8 secs
household problem simulated along transition in 0.1 secs

['tau']
household problem solved along transition path in 0.8 secs
household problem simulated along transition in 0.1 secs
```





[42]:
```
from HANKModel import RANKModelClass
```

[43]:
```
model_RA = RANKModelClass(name='RA')
model_RA.find_ss()
model_RA.test_path()
```

```
shocks: Gamma istar G
unknowns: Y w pi

look at max(abs(path.VARNAME[:]-ss.VARNAME)):
```

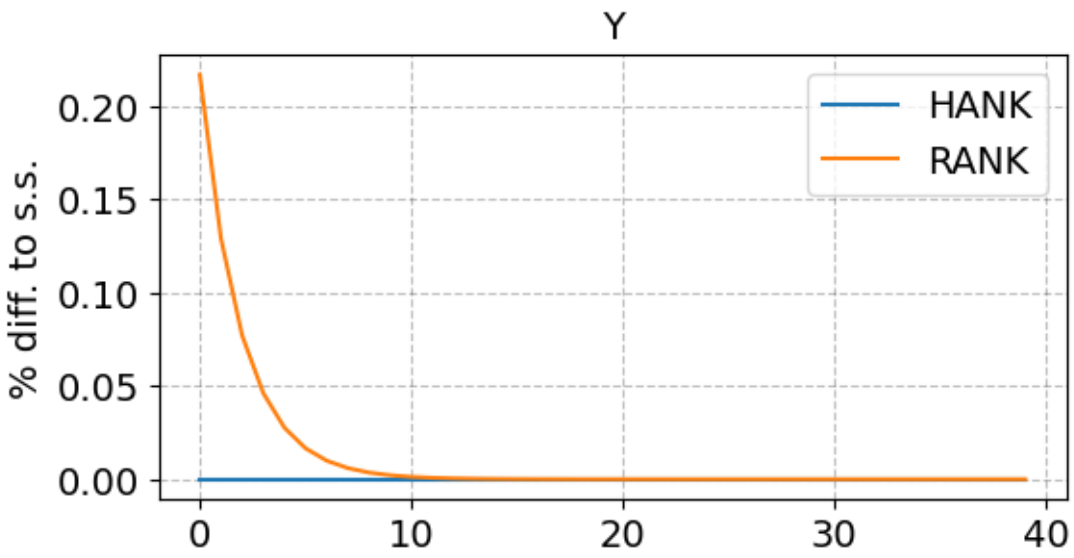
```
blocks.production
 N                 0.0e+00
 s                 0.0e+00
blocks.taylor
 i                 0.0e+00
blocks.fisher
 r                 8.7e-18
blocks.government
 B                 0.0e+00
 tau               4.9e-17
blocks.intermediary_goods
 NKPC_res          0.0e+00 [target]
 adjcost           0.0e+00
 d                 0.0e+00
hh
 A_hh              2.3e-09
 C_hh              5.2e-11
 ELL_hh            5.1e-11
 N_hh              4.2e-11
blocks.market_clearing
 A                 0.0e+00
 clearing_N        1.3e-10 [target]
 clearing_A        1.3e-10 [target]
 clearing_Y        6.4e-11
```

[44]:
```
model_RA.compute_jacs(do_print=False)
```

[45]:
```
model_RA.find_transition_path(shocks=['istar'],do_print=False)
```

[119]:
```
# Set the DPI to 2000
plt.figure(dpi=2000)
model.
 ↪compare_IRFs([model,model_RA],['HANK','RANK'],['Y'],do_shocks=False,do_targets=False,ncols=
# Save the figure directly using savefig without displaying
plt.savefig(r"C:\Users\USER\Documents\GitHub\hank sticky\hankvsrank.png",␣
 ↪dpi=500)
```

```
<Figure size 12800x9600 with 0 Axes>
```

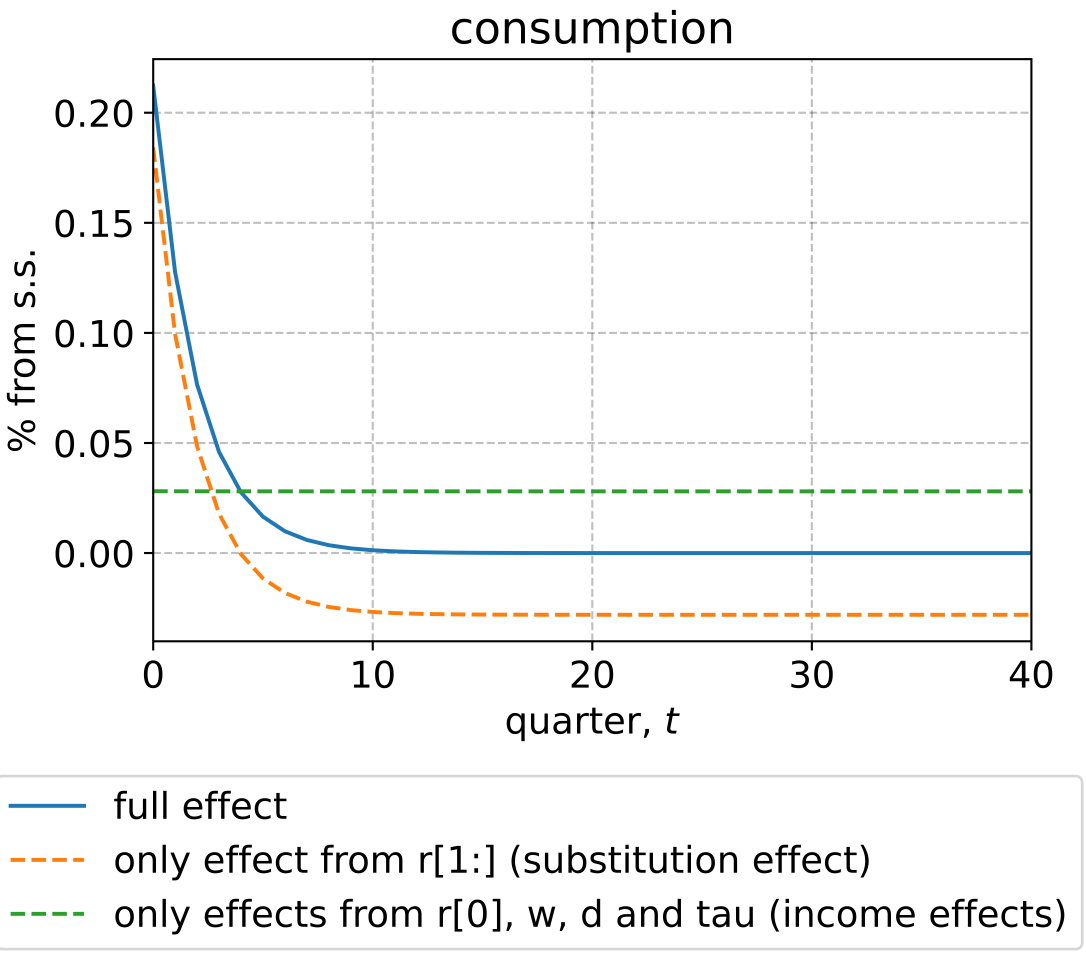



```
<Figure size 640x480 with 0 Axes>
```

[47]:
```
assert np.isclose(ss.r*ss.A_hh+ss.w*ss.N+ss.d-ss.tau,ss.C_hh) # flow budget
assert np.isclose(ss.C_hh,ss.Y-ss.G-ss.adjcost) # resource constraint

# IBC
q = np.ones((par.T,1))
for t in range(1,par.T): q[t] = q[t-1]*1/(1+ss.r)

C_NPV = np.sum(q*ss.C_hh)
Y_RA_NPV = (1+ss.r)*ss.A_hh + np.sum(q*(ss.w*ss.N+ss.d-ss.tau))

assert np.isclose(C_NPV,Y_RA_NPV)
```

[49]:
```
fig = plt.figure(figsize=(6,4),dpi=2000)
ax = fig.add_subplot(1,1,1)
ax.set_title('consumption')

# a. full effect
C_RA = model_RA.path.Y-model_RA.path.G-model_RA.path.adjcost
ax.plot((C_RA/ss.C_hh-1)*100,label='full effect')

# bI. relative path
C_RA_r = np.zeros_like(C_RA)
for k in range(par.T): # backwards
    t = par.T-1-k
    C_RA_plus = C_RA_r[t+1] if t+1 < par.T else ss.C_hh
```

```
    r_plus = model_RA.path.r[t+1] if t+1 < par.T else ss.r
    C_RA_r[t] = (par.beta_RA*(1+r_plus))**(-1/par.sigma)*C_RA_plus # Euler

# bII. scale to satisfy IBC
q = np.ones((par.T,1))
for t in range(1,par.T): q[t] = q[t-1]*1/(1+model_RA.path.r[t])

C_RA_r_NPV = np.sum(q*C_RA_r)
Y_RA_NPV = (1+model_RA.ss.r)*ss.A_hh + np.sum(q*(ss.w*ss.N+ss.d-ss.tau))
fac = Y_RA_NPV/C_RA_r_NPV
C_RA_r *= fac

# bIII. plot only effect from r[1:]
ax.plot((C_RA_r/ss.C_hh-1)*100,ls='--',label='only effect from r[1:]␣
 ↪(substitution effect)')
ax.set_xlim([0,40])

# c. other effects
ax.plot((C_RA-C_RA_r)/ss.C_hh*100,ls='--',label='only effects from r[0], w, d␣
 ↪and tau (income effects)')
ax.set_xlim([0,40])

ax.set_xlabel('quarter, $t$')
ax.set_ylabel('% from s.s.')
ax.legend(frameon=True,ncol=1,bbox_to_anchor=(-0.2,-0.2), loc='upper left');
```

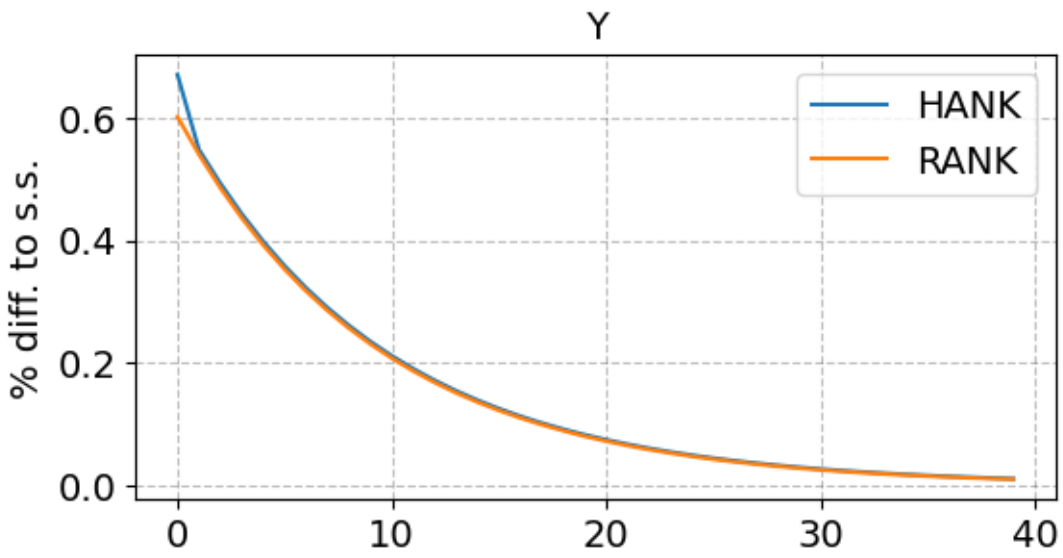


[50]:
```
model_G = model.copy()
model_G.find_transition_path(shocks=['G'],do_print=False)
```

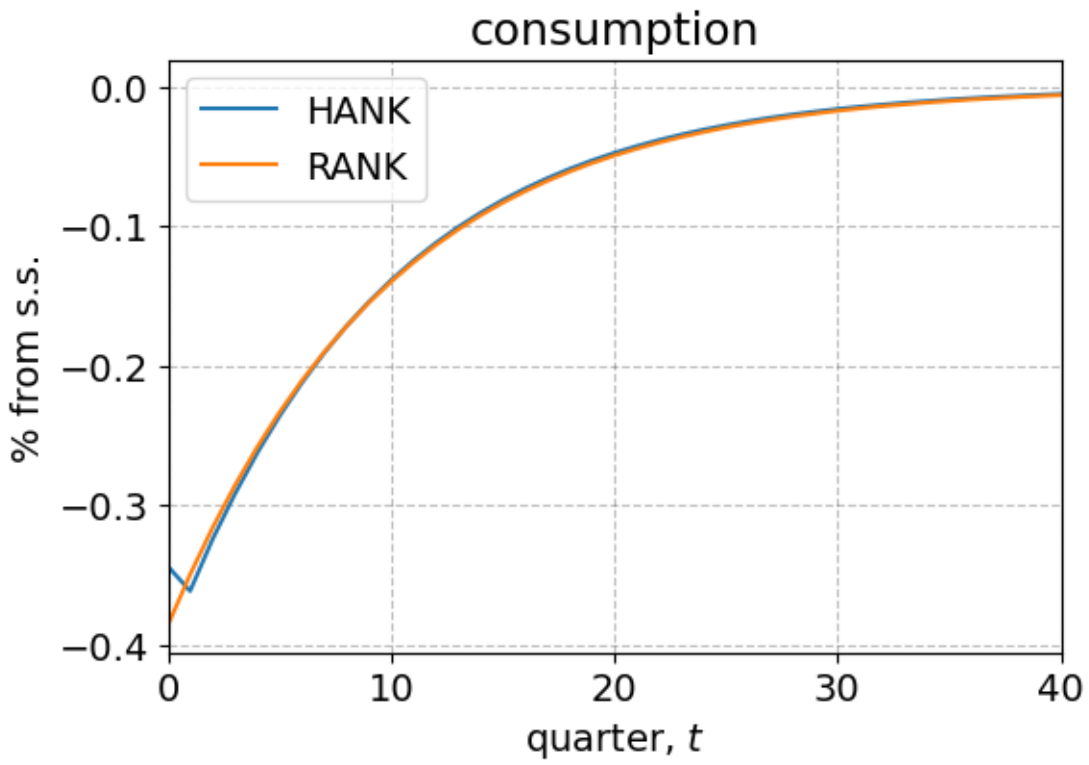
[51]:
```
model_RA_G = model_RA.copy()
model_RA_G.find_transition_path(shocks=['G'],do_print=False)
```

[60]:
```
# Set the DPI to 2000
plt.figure(dpi=1000)

model_G.
 ↪compare_IRFs([model_G,model_RA_G],['HANK','RANK'],['Y'],do_shocks=False,do_targets=False,nc
```

```
<Figure size 6400x4800 with 0 Axes>
```



[56]:
```
fig = plt.figure(figsize=(6,4))
ax = fig.add_subplot(1,1,1)
ax.set_title('consumption')

ax.plot((model_G.path.C_hh/ss.C_hh-1)*100,label='HANK')

C_RA = model_RA_G.path.Y-model_RA_G.path.G--model_RA_G.path.adjcost
ax.plot((C_RA/ss.C_hh-1)*100,label='RANK')

ax.set_xlim([0,40])
ax.set_xlabel('quarter, $t$')
ax.set_ylabel('% from s.s.')
ax.legend(frameon=True);
```

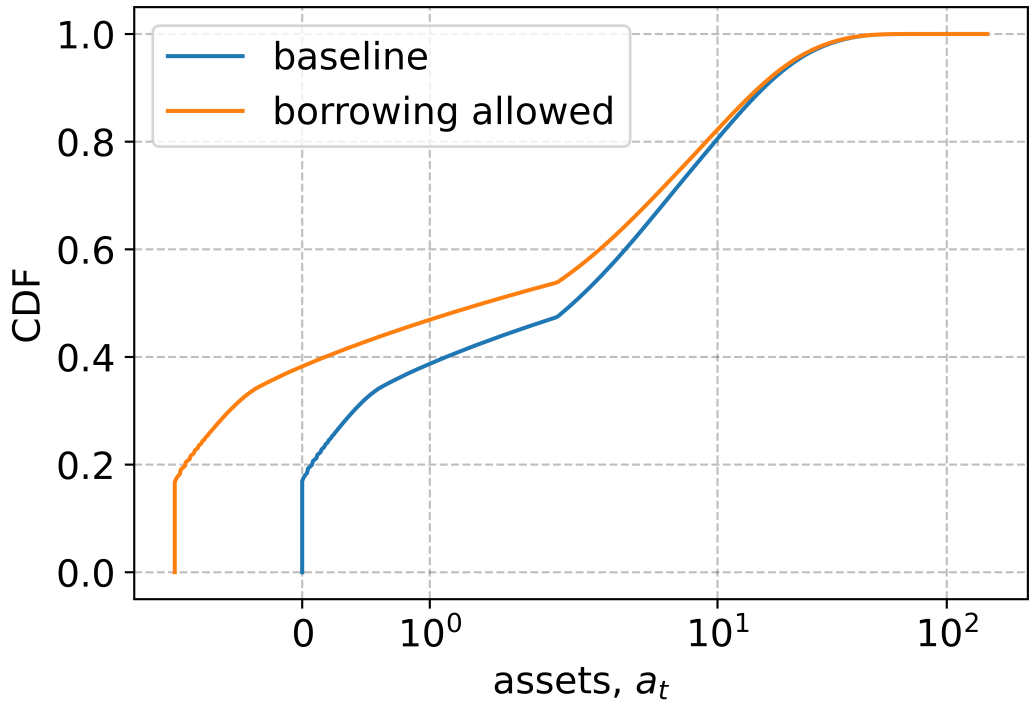


[57]:
```
model_ = model.copy(name='borrowing allowed')
model_.par.a_min = -1.0
model_.solve_hh_ss()
model_.simulate_hh_ss()
```

[59]:
```
fig = plt.figure(figsize=(6,4),dpi=1000)
ax = fig.add_subplot(1,1,1)

for m in [model,model_]:

    print(m.name)
    par = m.par
    ss = m.ss

    print(f'{ss.A_hh = :.2f}')
    print(f'{np.sum(ss.c**(1-par.sigma)/(1-par.sigma)*ss.D) = :.4f}')

    y = np.insert(np.cumsum(np.sum(ss.D[0],axis=0)),0,0.0)
    ax.plot(np.insert(par.a_grid,0,par.a_grid[0]),y/y[-1],label=m.name)
    print('')

ax.legend()
```

```
ax.set_xlabel('assets, $a_{t}$')
ax.set_ylabel('CDF')
ax.set_xscale('symlog')
```

```
baseline
ss.A_hh = 5.60
np.sum(ss.c**(1-par.sigma)/(1-par.sigma)*ss.D) = -1.1310

borrowing allowed
ss.A_hh = 4.70
np.sum(ss.c**(1-par.sigma)/(1-par.sigma)*ss.D) = -1.1380
```



[120]:
```
#Calibrated Param
print(model.par)
```

```
namespace(RA=False, Nfix=1, r_target_ss=0.01, beta=0.9773034960315107,
varphi=0.7888500056593879, sigma=2.0, nu=2.0, beta_RA=0.9900990099009901,
varphi_RA=0.8333333334406864, rho_z=0.965, sigma_psi=0.1311249404194336, Nz=7,
mu=1.2, kappa=0.05, phi=1.5, phi_y=0.0, G_target_ss=0.0, B_target_ss=5.6,
a_min=0.0, a_max=150.0, Na=500, jump_Gamma=0.01, rho_Gamma=0.9, std_Gamma=0.0,
jump_istar=-0.0025, rho_istar=0.6, std_istar=0.0025, jump_G=0.01, rho_G=0.9,
std_G=0.0, T=1000, max_iter_solve=50000, max_iter_simulate=50000,
max_iter_broyden=100, tol_ss=1e-12, tol_solve=1e-12, tol_simulate=1e-12,
tol_broyden=1e-10, py_hh=False, py_block=True, full_z_trans=False, simT=1000,
```

```
py_blocks=True, warnings=True, a_grid=array([0.00000000e+00, 3.22635016e-03,
6.49433765e-03, 9.80449984e-03,
       1.31573810e-02, 1.65535324e-02, 1.99935125e-02, 2.34778870e-02,
       2.70072287e-02, 3.05821179e-02, 3.42031426e-02, 3.78708980e-02,
       4.15859873e-02, 4.53490213e-02, 4.91606187e-02, 5.30214063e-02,
       5.69320190e-02, 6.08930996e-02, 6.49052996e-02, 6.89692787e-02,
       7.30857050e-02, 7.72552555e-02, 8.14786157e-02, 8.57564800e-02,
       9.00895519e-02, 9.44785438e-02, 9.89241774e-02, 1.03427184e-01,
       1.07988303e-01, 1.12608286e-01, 1.17287891e-01, 1.22027888e-01,
       1.26829057e-01, 1.31692187e-01, 1.36618078e-01, 1.41607539e-01,
       1.46661391e-01, 1.51780465e-01, 1.56965603e-01, 1.62217657e-01,
       1.67537491e-01, 1.72925980e-01, 1.78384009e-01, 1.83912476e-01,
       1.89512291e-01, 1.95184373e-01, 2.00929655e-01, 2.06749083e-01,
       2.12643613e-01, 2.18614214e-01, 2.24661869e-01, 2.30787570e-01,
       2.36992326e-01, 2.43277157e-01, 2.49643097e-01, 2.56091191e-01,
       2.62622501e-01, 2.69238099e-01, 2.75939075e-01, 2.82726530e-01,
       2.89601579e-01, 2.96565353e-01, 3.03618998e-01, 3.10763673e-01,
       3.18000553e-01, 3.25330828e-01, 3.32755703e-01, 3.40276398e-01,
       3.47894152e-01, 3.55610215e-01, 3.63425858e-01, 3.71342364e-01,
       3.79361036e-01, 3.87483193e-01, 3.95710169e-01, 4.04043317e-01,
       4.12484008e-01, 4.21033630e-01, 4.29693587e-01, 4.38465306e-01,
       4.47350226e-01, 4.56349810e-01, 4.65465537e-01, 4.74698907e-01,
       4.84051437e-01, 4.93524664e-01, 5.03120148e-01, 5.12839465e-01,
       5.22684214e-01, 5.32656014e-01, 5.42756503e-01, 5.52987343e-01,
       5.63350217e-01, 5.73846827e-01, 5.84478900e-01, 5.95248185e-01,
       6.06156451e-01, 6.17205493e-01, 6.28397128e-01, 6.39733195e-01,
       6.51215558e-01, 6.62846106e-01, 6.74626750e-01, 6.86559429e-01,
       6.98646104e-01, 7.10888762e-01, 7.23289416e-01, 7.35850106e-01,
       7.48572896e-01, 7.61459880e-01, 7.74513175e-01, 7.87734928e-01,
       8.01127313e-01, 8.14692532e-01, 8.28432815e-01, 8.42350423e-01,
       8.56447643e-01, 8.70726793e-01, 8.85190221e-01, 8.99840306e-01,
       9.14679456e-01, 9.29710111e-01, 9.44934742e-01, 9.60355854e-01,
       9.75975981e-01, 9.91797692e-01, 1.00782359e+00, 1.02405631e+00,
       1.04049851e+00, 1.05715291e+00, 1.07402225e+00, 1.09110928e+00,
       1.10841684e+00, 1.12594775e+00, 1.14370491e+00, 1.16169123e+00,
       1.17990967e+00, 1.19836322e+00, 1.21705493e+00, 1.23598786e+00,
       1.25516513e+00, 1.27458989e+00, 1.29426533e+00, 1.31419470e+00,
       1.33438126e+00, 1.35482833e+00, 1.37553928e+00, 1.39651752e+00,
       1.41776649e+00, 1.43928968e+00, 1.46109064e+00, 1.48317295e+00,
       1.50554024e+00, 1.52819619e+00, 1.55114453e+00, 1.57438902e+00,
       1.59793349e+00, 1.62178181e+00, 1.64593791e+00, 1.67040575e+00,
       1.69518935e+00, 1.72029280e+00, 1.74572022e+00, 1.77147579e+00,
       1.79756374e+00, 1.82398837e+00, 1.85075402e+00, 1.87786509e+00,
       1.90532605e+00, 1.93314139e+00, 1.96131571e+00, 1.98985362e+00,
       2.01875983e+00, 2.04803908e+00, 2.07769620e+00, 2.10773605e+00,
       2.13816358e+00, 2.16898379e+00, 2.20020174e+00, 2.23182258e+00,
       2.26385149e+00, 2.29629375e+00, 2.32915469e+00, 2.36243972e+00,
       2.39615430e+00, 2.43030398e+00, 2.46489438e+00, 2.49993118e+00,
```

2.53542014e+00, 2.57136710e+00, 2.60777798e+00, 2.64465875e+00,
2.68201548e+00, 2.71985431e+00, 2.75818147e+00, 2.79700326e+00,
2.83632606e+00, 2.87615633e+00, 2.91650063e+00, 2.95736559e+00,
2.99875793e+00, 3.04068445e+00, 3.08315205e+00, 3.12616771e+00,
3.16973851e+00, 3.21387161e+00, 3.25857426e+00, 3.30385381e+00,
3.34971772e+00, 3.39617352e+00, 3.44322885e+00, 3.49089145e+00,
3.53916915e+00, 3.58806990e+00, 3.63760173e+00, 3.68777278e+00,
3.73859132e+00, 3.79006569e+00, 3.84220435e+00, 3.89501589e+00,
3.94850898e+00, 4.00269242e+00, 4.05757512e+00, 4.11316610e+00,
4.16947451e+00, 4.22650960e+00, 4.28428075e+00, 4.34279746e+00,
4.40206935e+00, 4.46210617e+00, 4.52291779e+00, 4.58451420e+00,
4.64690555e+00, 4.71010207e+00, 4.77411418e+00, 4.83895238e+00,
4.90462735e+00, 4.97114988e+00, 5.03853091e+00, 5.10678152e+00,
5.17591294e+00, 5.24593652e+00, 5.31686378e+00, 5.38870639e+00,
5.46147615e+00, 5.53518504e+00, 5.60984517e+00, 5.68546882e+00,
5.76206842e+00, 5.83965657e+00, 5.91824603e+00, 5.99784972e+00,
6.07848072e+00, 6.16015230e+00, 6.24287788e+00, 6.32667107e+00,
6.41154565e+00, 6.49751557e+00, 6.58459496e+00, 6.67279814e+00,
6.76213963e+00, 6.85263410e+00, 6.94429644e+00, 7.03714171e+00,
7.13118520e+00, 7.22644235e+00, 7.32292883e+00, 7.42066051e+00,
7.51965346e+00, 7.61992395e+00, 7.72148847e+00, 7.82436372e+00,
7.92856662e+00, 8.03411430e+00, 8.14102412e+00, 8.24931364e+00,
8.35900069e+00, 8.47010329e+00, 8.58263972e+00, 8.69662847e+00,
8.81208830e+00, 8.92903818e+00, 9.04749734e+00, 9.16748527e+00,
9.28902169e+00, 9.41212659e+00, 9.53682020e+00, 9.66312304e+00,
9.79105586e+00, 9.92063971e+00, 1.00518959e+01, 1.01848460e+01,
1.03195119e+01, 1.04559156e+01, 1.05940798e+01, 1.07340270e+01,
1.08757802e+01, 1.10193629e+01, 1.11647985e+01, 1.13121111e+01,
1.14613247e+01, 1.16124641e+01, 1.17655539e+01, 1.19206195e+01,
1.20776862e+01, 1.22367799e+01, 1.23979268e+01, 1.25611534e+01,
1.27264865e+01, 1.28939532e+01, 1.30635812e+01, 1.32353983e+01,
1.34094328e+01, 1.35857132e+01, 1.37642686e+01, 1.39451284e+01,
1.41283222e+01, 1.43138802e+01, 1.45018329e+01, 1.46922112e+01,
1.48850465e+01, 1.50803703e+01, 1.52782149e+01, 1.54786127e+01,
1.56815967e+01, 1.58872004e+01, 1.60954574e+01, 1.63064021e+01,
1.65200691e+01, 1.67364936e+01, 1.69557111e+01, 1.71777577e+01,
1.74026698e+01, 1.76304846e+01, 1.78612394e+01, 1.80949722e+01,
1.83317215e+01, 1.85715260e+01, 1.88144254e+01, 1.90604594e+01,
1.93096686e+01, 1.95620940e+01, 1.98177770e+01, 2.00767597e+01,
2.03390847e+01, 2.06047951e+01, 2.08739345e+01, 2.11465474e+01,
2.14226784e+01, 2.17023730e+01, 2.19856772e+01, 2.22726375e+01,
2.25633011e+01, 2.28577159e+01, 2.31559303e+01, 2.34579932e+01,
2.37639543e+01, 2.40738640e+01, 2.43877732e+01, 2.47057336e+01,
2.50277973e+01, 2.53540174e+01, 2.56844475e+01, 2.60191419e+01,
2.63581557e+01, 2.67015446e+01, 2.70493651e+01, 2.74016744e+01,
2.77585303e+01, 2.81199916e+01, 2.84861177e+01, 2.88569688e+01,
2.92326059e+01, 2.96130908e+01, 2.99984859e+01, 3.03888547e+01,
3.07842614e+01, 3.11847710e+01, 3.15904493e+01, 3.20013631e+01,

```
       3.24175798e+01, 3.28391681e+01, 3.32661970e+01, 3.36987370e+01,
       3.41368590e+01, 3.45806352e+01, 3.50301385e+01, 3.54854428e+01,
       3.59466230e+01, 3.64137550e+01, 3.68869154e+01, 3.73661822e+01,
       3.78516341e+01, 3.83433509e+01, 3.88414136e+01, 3.93459039e+01,
       3.98569050e+01, 4.03745006e+01, 4.08987761e+01, 4.14298175e+01,
       4.19677123e+01, 4.25125488e+01, 4.30644166e+01, 4.36234065e+01,
       4.41896104e+01, 4.47631213e+01, 4.53440337e+01, 4.59324430e+01,
       4.65284459e+01, 4.71321405e+01, 4.77436260e+01, 4.83630030e+01,
       4.89903732e+01, 4.96258400e+01, 5.02695077e+01, 5.09214822e+01,
       5.15818707e+01, 5.22507817e+01, 5.29283253e+01, 5.36146129e+01,
       5.43097573e+01, 5.50138729e+01, 5.57270753e+01, 5.64494819e+01,
       5.71812114e+01, 5.79223842e+01, 5.86731221e+01, 5.94335486e+01,
       6.02037887e+01, 6.09839691e+01, 6.17742180e+01, 6.25746653e+01,
       6.33854428e+01, 6.42066837e+01, 6.50385230e+01, 6.58810976e+01,
       6.67345459e+01, 6.75990083e+01, 6.84746269e+01, 6.93615458e+01,
       7.02599107e+01, 7.11698693e+01, 7.20915713e+01, 7.30251683e+01,
       7.39708137e+01, 7.49286630e+01, 7.58988738e+01, 7.68816055e+01,
       7.78770198e+01, 7.88852803e+01, 7.99065528e+01, 8.09410052e+01,
       8.19888076e+01, 8.30501324e+01, 8.41251540e+01, 8.52140491e+01,
       8.63169969e+01, 8.74341787e+01, 8.85657782e+01, 8.97119814e+01,
       9.08729768e+01, 9.20489553e+01, 9.32401103e+01, 9.44466376e+01,
       9.56687357e+01, 9.69066054e+01, 9.81604503e+01, 9.94304766e+01,
       1.00716893e+02, 1.02019911e+02, 1.03339745e+02, 1.04676613e+02,
       1.06030733e+02, 1.07402328e+02, 1.08791624e+02, 1.10198850e+02,
       1.11624237e+02, 1.13068019e+02, 1.14530433e+02, 1.16011721e+02,
       1.17512125e+02, 1.19031892e+02, 1.20571273e+02, 1.22130520e+02,
       1.23709889e+02, 1.25309641e+02, 1.26930039e+02, 1.28571348e+02,
       1.30233839e+02, 1.31917786e+02, 1.33623464e+02, 1.35351154e+02,
       1.37101142e+02, 1.38873713e+02, 1.40669160e+02, 1.42487779e+02,
       1.44329867e+02, 1.46195728e+02, 1.48085669e+02, 1.50000000e+02]),
z_grid=array([0.25952913, 0.39037867, 0.58720002, 0.88325488, 1.32857484,
       1.99841649, 3.00597929]))
```

[ ]: