In [ ]:
```
import numpy as np
import numpy.linalg as la
import numpy.linalg as la
import matplotlib.pyplot as plt
from solve_hand_to_mouth import *
from copy import deepcopy
from estimation import *

%load_ext autoreload
%autoreload 2

# %pip install EconModel
from Model import ReferenceDependenceClass
model = ReferenceDependenceClass()
model.par.full_sample_estimation = True
model.allocate()
model_standard = deepcopy(model)    # Used later for standard model
model.par.model = 'HTM'
model_standard.par.model = 'HTM'
```

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

Estimating using the full sample

(using hazard rates from both before and after the reform)

Model with reference dependence

In [ ]:
```
# Setup
model.par.eta = 1  # Reference dependence
model.par.types = 1  # No heterogeneity
model.allocate()

# Estimation setup
est_par = ['delta', 'gamma', 'lambdaa', 'N', 'cost1'] # Parameters to estimate
theta0 = [0.95, 0.1, 1.0, 10, 100.0]  # Initial guesses
bounds = [(0.5, 1), (0.001, 10.0), (0.0, 10.0), (0,50), (0.0,500)] # Bounds for


#  Random starting values:
model.par.noOfParams = len(est_par)
model.par.noSearchInits = 800
np.random.seed(180615)

est_best = np.inf

for i in range(model.par.noSearchInits):

    theta0 = [np.random.uniform(model.par.lb_delta, model.par.ub_delta),
              np.random.uniform(model.par.lb_gamma,model.par.ub_gamma),
              np.random.uniform(model.par.lb_lambdaa,model.par.ub_lambdaa),
              np.random.uniform(model.par.lb_N,model.par.ub_N),
              np.random.uniform(model.par.lb_c,model.par.ub_c)]
```

```
    try:
        # Perform the method of simulated moments
        est = method_simulated_moments(model, est_par, theta0, bounds, weight=Fa
        # print(est.fun)

        # Check if the current estimate is better than the best estimate
        if est.fun < est_best:
            est_best = est.fun
            par = est.x
            nit = est.nit
            jac = est.jac
            obj_ref = est_best

    except Exception as e:
        # Print the exception message and continue with the next iteration
        print(f"Iteration {i} failed with error: {e}")
        continue
```

```
c:\Users\rasmu\anaconda3\Lib\site-packages\scipy\optimize\_optimize.py:404: Runti
meWarning: Values in x were outside bounds during a minimize step, clipping to bo
unds
  warnings.warn("Values in x were outside bounds during a "
d:\OneDrive\KU - Økonomi\Dynamic Programming\Term_Paper\Dynamic-programming-proje
ct\Funcs.py:64: RuntimeWarning: invalid value encountered in scalar power
  inv_c_marg[0] = (s/par.cost1)**(1/par.gamma)
d:\OneDrive\KU - Økonomi\Dynamic Programming\Term_Paper\Dynamic-programming-proje
ct\Funcs.py:65: RuntimeWarning: invalid value encountered in scalar power
  inv_c_marg[1] = (s/par.cost2)**(1/par.gamma)
d:\OneDrive\KU - Økonomi\Dynamic Programming\Term_Paper\Dynamic-programming-proje
ct\Funcs.py:66: RuntimeWarning: invalid value encountered in scalar power
  inv_c_marg[2] = (s/par.cost3)**(1/par.gamma)
```

In [ ]:

```
# Calculates jacobian based on small changes (epsilon)
def manual_jacobian(model, est_par, par, epsilon=1e-5):
    '''Calculate the Jacobian using finite differences'''
    baseline_moments = model_moments_combined(model, est_par, par)
    num_moments = len(baseline_moments)
    num_params = len(est_par)

    jacobian = np.zeros((num_moments, num_params))  # Set up empty matrix to sto

    for i in range(num_params): # Loop over the parameters
        new_par = np.array(par) # Copy the parameter vector
        new_par[i] += epsilon   # Add a small perturbation to the i-th parameter
        perturbed_moments = model_moments_combined(model, est_par, new_par) # Ca
        jacobian[:, i] = (perturbed_moments - baseline_moments) / epsilon   # Ca
        new_par = np.array(par) # Reset vector to the original parameter vector

    return jacobian

jacobian = manual_jacobian(model, est_par, par)

print(jacobian)
print(jacobian.shape)
```

```
[[ 3.29088178e-01  1.61272204e-01  8.80951060e-03  0.00000000e+00
  -1.80021752e-04]
 [ 3.14500326e-01  1.55721063e-01  8.05255910e-03  0.00000000e+00
  -1.70325478e-04]
 [ 3.00810822e-01  1.49758590e-01  7.28034123e-03  0.00000000e+00
  -1.60387213e-04]
 [ 2.88310958e-01  1.43445279e-01  6.50187640e-03  0.00000000e+00
  -1.50332747e-04]
 [ 2.77301663e-01  1.36879025e-01  5.72852946e-03  0.00000000e+00
  -1.40326449e-04]
 [ 2.68085389e-01  1.30204011e-01  4.97418887e-03  0.00000000e+00
  -1.30576520e-04]
 [ 2.60958789e-01  1.23620472e-01  4.25544196e-03  0.00000000e+00
  -1.21340937e-04]
 [ 2.56207610e-01  1.17395269e-01  3.59182482e-03  0.00000000e+00
  -1.12935104e-04]
 [ 2.54104564e-01  1.11873228e-01  3.00632193e-03  0.00000000e+00
  -1.05743388e-04]
 [ 2.54908651e-01  1.07489409e-01  2.52646512e-03  0.00000000e+00
  -1.00238384e-04]
 [ 2.58859410e-01  1.04782890e-01  2.18666773e-03  0.00000000e+00
  -9.70147213e-05]
 [ 2.66151755e-01  1.04413383e-01  2.03285846e-03  0.00000000e+00
  -9.68482093e-05]
 [ 2.76868941e-01  1.07182926e-01  2.13106795e-03  0.00000000e+00
  -1.00796418e-04]
 [ 2.90849591e-01  1.14065283e-01  2.58247844e-03  0.00000000e+00
  -1.10362425e-04]
 [ 3.04816326e-01  1.22456239e-01  3.14601029e-03  0.00000000e+00
  -1.22384315e-04]
 [ 3.18180507e-01  1.32747095e-01  3.85609318e-03  0.00000000e+00
  -1.37630500e-04]
 [ 3.30045214e-01  1.45466013e-01  4.76071400e-03  0.00000000e+00
  -1.57176758e-04]
 [ 3.39047389e-01  1.61345372e-01  5.92838582e-03  0.00000000e+00
  -1.82565623e-04]
 [ 3.30295223e-01  1.59284429e-01  5.57083059e-03  0.00000000e+00
  -1.78606403e-04]
 [ 3.21734042e-01  1.57492249e-01  5.23352423e-03  0.00000000e+00
  -1.75143277e-04]
 [ 3.13318823e-01  1.56106404e-01  4.92499388e-03  0.00000000e+00
  -1.72368801e-04]
 [ 3.04947948e-01  1.55302712e-01  4.65570693e-03  0.00000000e+00
  -1.70525622e-04]
 [ 2.96437875e-01  1.55304593e-01  4.43860204e-03  0.00000000e+00
  -1.69921056e-04]
 [ 2.87486716e-01  1.56395310e-01  4.28986925e-03  0.00000000e+00
  -1.70947784e-04]
 [ 2.73371615e-01  1.50701082e-01  3.61784626e-03  0.00000000e+00
  -1.61354428e-04]
 [ 2.60427393e-01  1.44842767e-01  2.96482988e-03  0.00000000e+00
  -1.51897513e-04]
 [ 2.48889158e-01  1.38973425e-01  2.34480544e-03  0.00000000e+00
  -1.42796882e-04]
 [ 2.38965648e-01  1.33300745e-01  1.77414251e-03  0.00000000e+00
  -1.34322462e-04]
 [ 2.30822558e-01  1.28095030e-01  1.27191432e-03  0.00000000e+00
  -1.26799788e-04]
 [ 2.24561272e-01  1.23694023e-01  8.60575765e-04  0.00000000e+00
  -1.20614604e-04]
```

```
 [ 2.20186873e-01  1.20496807e-01  5.67368787e-04  0.00000000e+00
  -1.16209591e-04]
 [ 2.17550106e-01  1.18915360e-01  4.27010276e-04  0.00000000e+00
  -1.14036171e-04]
 [ 2.15365383e-01  1.17464863e-01  3.00389477e-04  0.00000000e+00
  -1.12067312e-04]
 [ 2.13652072e-01  1.16190058e-01  1.90493464e-04  0.00000000e+00
  -1.10356619e-04]
 [ 2.12416456e-01  1.15143465e-01  1.00851395e-04  0.00000000e+00
  -1.08966538e-04]
 [ 3.35924832e-01  1.36569609e-01  7.05340423e-03  0.00000000e+00
  -1.43252277e-04]
 [ 3.39238434e-01  1.38871800e-01  7.06716384e-03  0.00000000e+00
  -1.46923580e-04]
 [ 3.43042651e-01  1.42773128e-01  7.19873449e-03  0.00000000e+00
  -1.53053511e-04]
 [ 3.46868533e-01  1.48726197e-01  7.48480759e-03  0.00000000e+00
  -1.62412105e-04]
 [ 3.49950604e-01  1.57320041e-01  7.97596231e-03  0.00000000e+00
  -1.76064055e-04]
 [ 3.51067363e-01  1.69341391e-01  8.74383037e-03  0.00000000e+00
  -1.95519895e-04]
 [ 3.35182706e-01  1.62821596e-01  7.81947760e-03  0.00000000e+00
  -1.83661686e-04]
 [ 3.20777558e-01  1.55987409e-01  6.90185408e-03  0.00000000e+00
  -1.71864008e-04]
 [ 3.08221033e-01  1.49048361e-01  6.01309837e-03  0.00000000e+00
  -1.60457254e-04]
 [ 2.97864249e-01  1.42306798e-01  5.18061074e-03  0.00000000e+00
  -1.49864166e-04]
 [ 2.90024387e-01  1.36178360e-01  4.43801041e-03  0.00000000e+00
  -1.40618531e-04]
 [ 2.84967610e-01  1.31214836e-01  3.82698820e-03  0.00000000e+00
  -1.33394461e-04]
 [ 2.82883970e-01  1.28129750e-01  3.40113690e-03  0.00000000e+00
  -1.29056012e-04]
 [ 2.83837612e-01  1.27826703e-01  3.23378990e-03  0.00000000e+00
  -1.28742669e-04]
 [ 2.86018865e-01  1.28877749e-01  3.16232495e-03  0.00000000e+00
  -1.30370003e-04]
 [ 2.89203416e-01  1.31684903e-01  3.21474723e-03  0.00000000e+00
  -1.34522662e-04]
 [ 2.92964272e-01  1.36746189e-01  3.42890201e-03  0.00000000e+00
  -1.41981026e-04]
 [ 2.96559654e-01  1.44686822e-01  3.85740260e-03  0.00000000e+00
  -1.53810954e-04]
 [ 2.91080389e-01  1.42350673e-01  3.52664744e-03  0.00000000e+00
  -1.50058237e-04]
 [ 2.87150645e-01  1.41767269e-01  3.37150093e-03  0.00000000e+00
  -1.48989100e-04]
 [ 2.83341170e-01  1.41711080e-01  3.24943265e-03  0.00000000e+00
  -1.48698076e-04]
 [ 2.79492426e-01  1.42342974e-01  3.16913795e-03  0.00000000e+00
  -1.49403377e-04]
 [ 2.75357451e-01  1.43863678e-01  3.14148627e-03  0.00000000e+00
  -1.51381752e-04]
 [ 2.70565735e-01  1.46524785e-01  3.18027317e-03  0.00000000e+00
  -1.54988231e-04]
 [ 2.60351360e-01  1.42326798e-01  2.71493474e-03  0.00000000e+00
  -1.48251161e-04]
```

```
[ 2.50956204e-01  1.38088768e-01  2.26504460e-03  0.00000000e+00
 -1.41660040e-04]
[ 2.42514317e-01  1.33902446e-01  1.83834050e-03  0.00000000e+00
 -1.35339821e-04]
[ 2.35144233e-01  1.29886395e-01  1.44382239e-03  0.00000000e+00
 -1.29441263e-04]
[ 2.28939133e-01  1.26189139e-01  1.09194548e-03  0.00000000e+00
 -1.24143463e-04]
[ 2.23953932e-01  1.22990457e-01  7.94942047e-04  0.00000000e+00
 -1.19655040e-04]
[ 2.20186873e-01  1.20496807e-01  5.67368787e-04  0.00000000e+00
 -1.16209591e-04]
[ 2.17550106e-01  1.18915360e-01  4.27010276e-04  0.00000000e+00
 -1.14036171e-04]
[ 2.15365383e-01  1.17464863e-01  3.00389477e-04  0.00000000e+00
 -1.12067312e-04]
[ 2.13652072e-01  1.16190058e-01  1.90493464e-04  0.00000000e+00
 -1.10356619e-04]
[ 2.12416456e-01  1.15143465e-01  1.00851395e-04  0.00000000e+00
 -1.08966538e-04]]
(70, 5)
```

In [ ]:

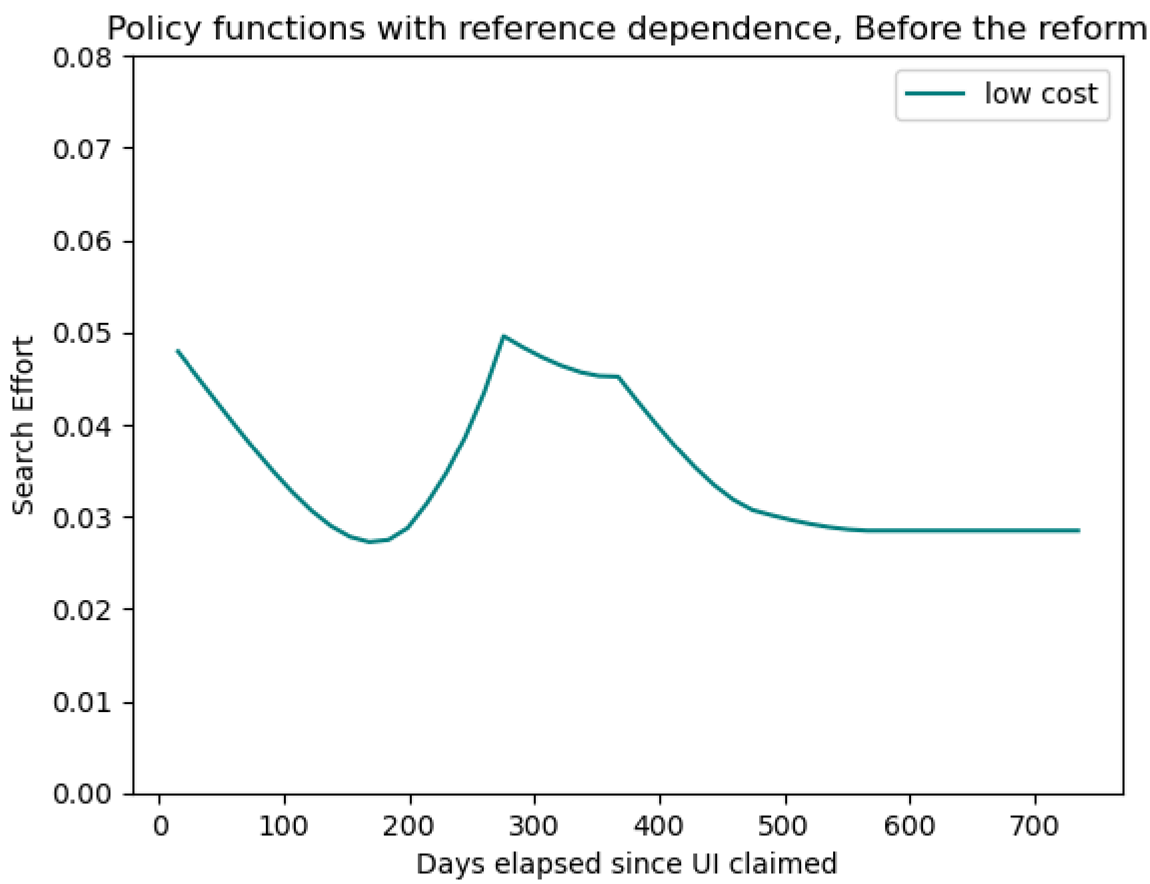
```
def standard_errors(jac, weight_matrix, N = 70):
    variance = np.zeros(jac.shape[1])*np.nan
    for i in range(jac.shape[1]):
        jac_i = jac[:, i].reshape(-1, 1)

        if np.all(jac_i == 0):
            print(f"Column {i} of the Jacobian is zero, cannot compute standard
            continue
        # diagonal Variance matirx for data moments
        omega_matrix = np.diag(np.diag(weight_matrix))

        # Compute (G'WG)
        gwg_inv = la.pinv(jac_i.T @ weight_matrix @ jac_i)

        # Compute the middle term G'WVW'G
        middle_term =  jac_i.T @ weight_matrix @ omega_matrix @ weight_matrix.T

        # Compute the variance
        variance[i] = gwg_inv @ middle_term @ gwg_inv

    standard_errors = np.sqrt(variance/N)

    return standard_errors

standard_errs = standard_errors(jacobian, model.data.weight_mat)

# print("Standard Errors:")
# print(standard_errs)


print("Optimization Results:")
print("---------------------")
print(f'{"Parameter":<15} {"Estimate":<15} {"Std. Error":<15}')
for param, estimate, error in zip(est_par, par, standard_errs):
    print(f'{param:<15} {estimate:>15.3f} {error:>15.3f}')
print(f'Objective:                {est_best:.4f}')
print(f'Number of iterations:     {nit}')
```

```
Column 3 of the Jacobian is zero, cannot compute standard errors.
Optimization Results:
---------------------
Parameter       Estimate        Std. Error
delta                   0.867           0.000
gamma                   0.799           0.000
lambdaa                 4.665           0.013
N                      13.834             nan
cost1                 302.578           0.434
Objective:             0.2672
Number of iterations:    16
```

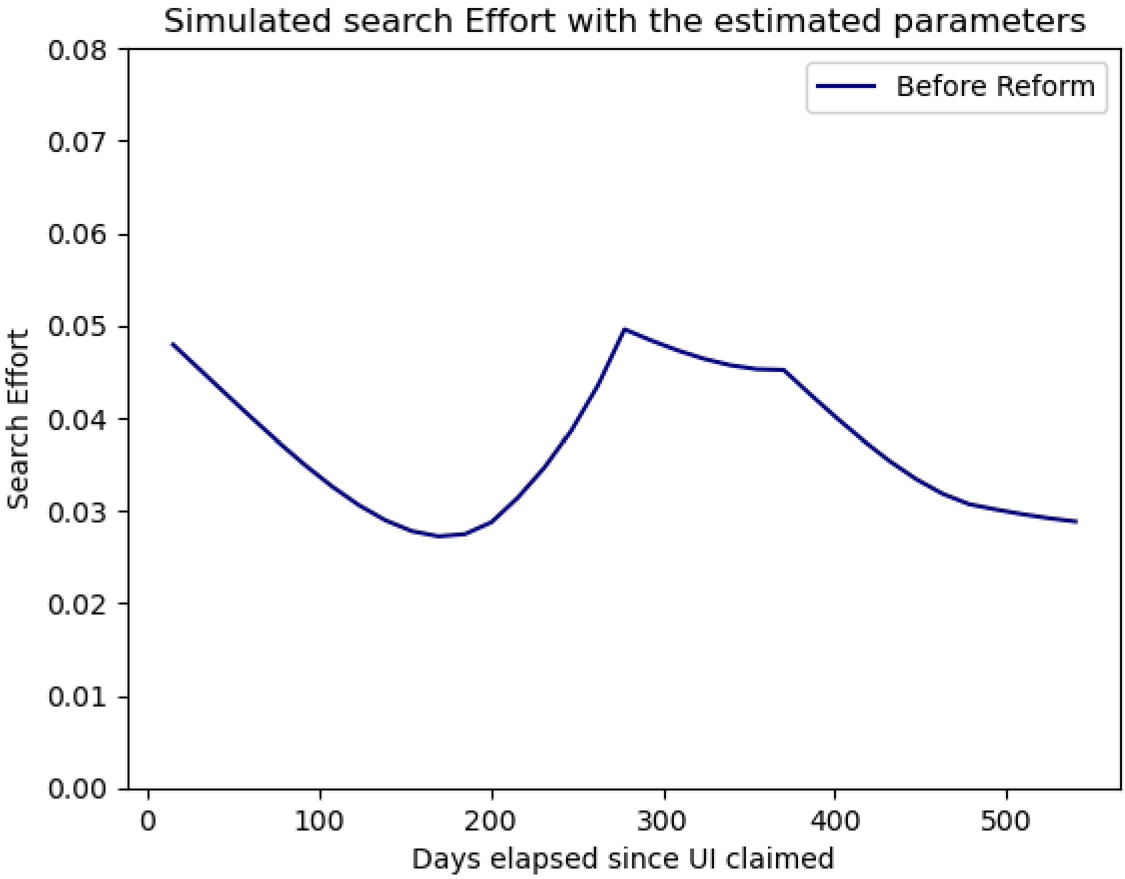
```
C:\Users\rasmu\AppData\Local\Temp\ipykernel_12552\3391549759.py:19: DeprecationWa
rning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will e
rror in future. Ensure you extract a single element from your array before perfor
ming this operation. (Deprecated NumPy 1.25.)
  variance[i] = gwg_inv @ middle_term @ gwg_inv
```

In [ ]:

```
# Plot policy functions
search_effort_reference = solve_search_effort_HTM(model.par)
time = np.linspace(0, model.par.T, model.par.T)
plt.plot((time+1)*15, search_effort_reference[0,:], label = 'low cost', color='t
# plt.plot((time+1)*15, search_effort[1,:], label = 'medium cost', color='orange
#plt.plot(time, search_effort[2,:], label = 'high' )
#plt.text(0.5, 0.96, '(Note that high cost is 0)', horizontalalignment='center',
plt.xlabel('Days elapsed since UI claimed')
plt.ylabel('Search Effort')
plt.title('Policy functions with reference dependence, Before the reform')
plt.legend()
plt.ylim(0.0, 0.08)
plt.show()
```

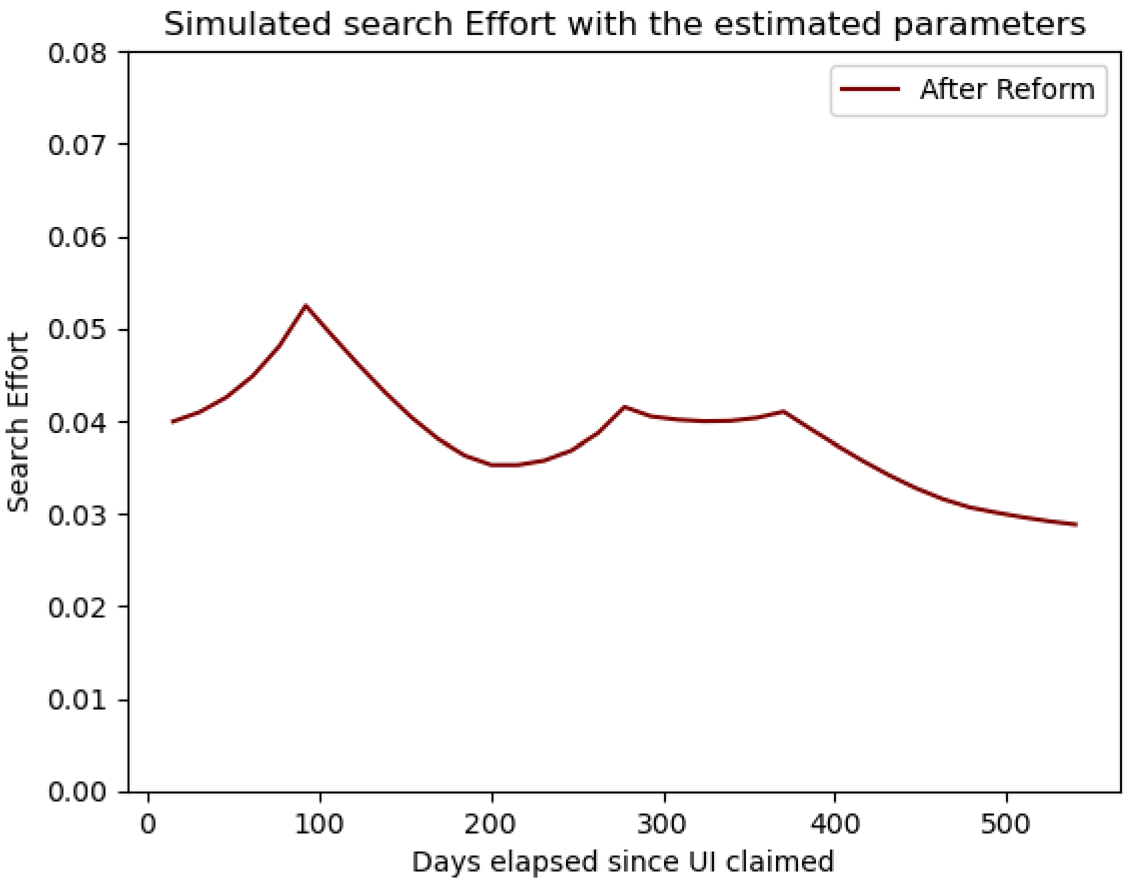


In [ ]:
```
# Policy function before reform
search_reference_beforeReform = sim_search_effort_HTM(model.par)
time = np.linspace(0, model.par.T_sim, model.par.T_sim)
plt.plot((time+1)*15, search_reference_beforeReform, color='navy', label='Before
plt.xlabel('Days elapsed since UI claimed')
plt.ylabel('Search Effort')
plt.title('Simulated search Effort with the estimated parameters')
plt.ylim(0.0, 0.08)
plt.legend()
plt.show()
```



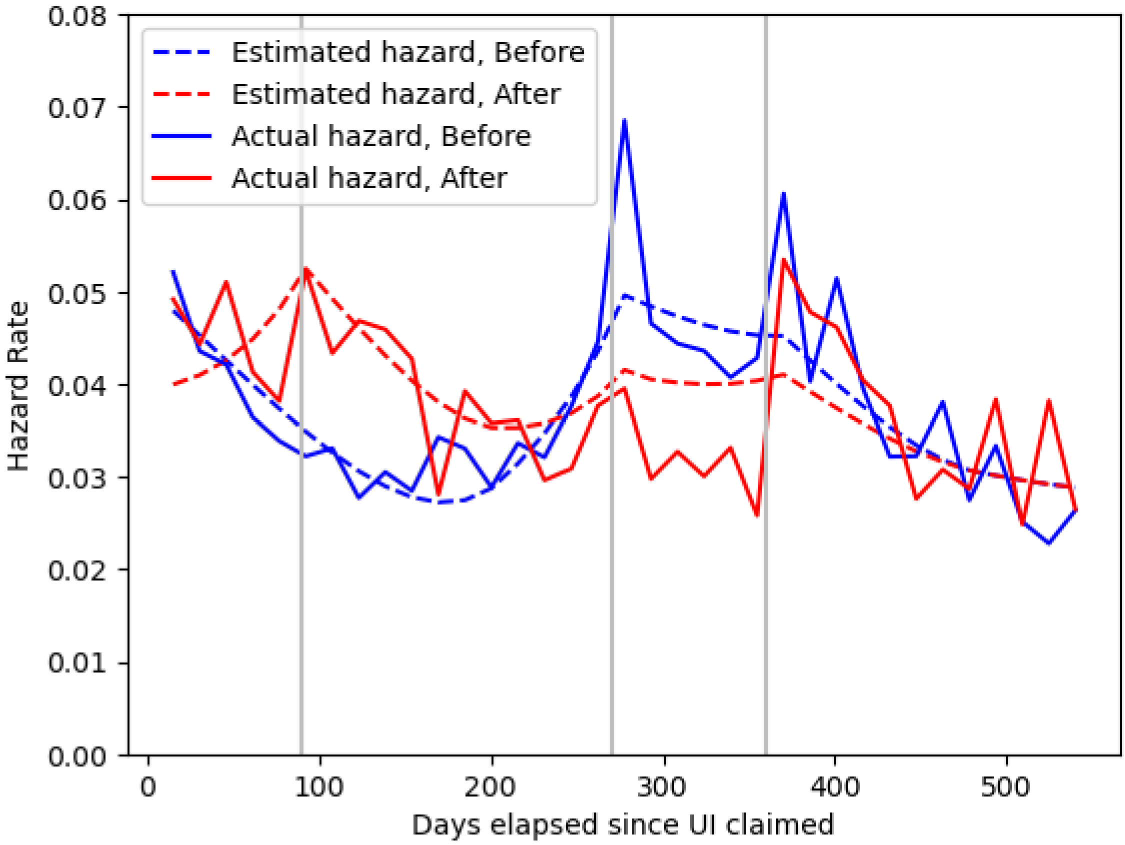
In [ ]:
```
# Policy function after reform
model_afterReform = deepcopy(model)
model_afterReform.par.b1 = 342.0/675.0      # Value after reform
model_afterReform.par.b2 = 171.0/675.0      # Value after reform

model_afterReform.allocate()
search_reference_afterReform = model_afterReform.solve()

# true_data_outofsample = model.data.moments_after
# true_data_insample = model.data.moments_before
# mse_eta1_outofsample = np.mean((true_data_outofsample - s_forecast)**2)
# mse_eta1_insample = np.mean((true_data_insample - sim)**2)

# Now plotting s_forecast
time = np.linspace(0, model.par.T_sim, model.par.T_sim)
plt.plot((time+1)*15, search_reference_afterReform, label='After Reform', color=
plt.xlabel('Days elapsed since UI claimed')
plt.ylabel('Search Effort')
plt.title('Simulated search Effort with the estimated parameters')
plt.legend()
```

```
plt.ylim(0.0, 0.08)
plt.show()
```



In [ ]:

```
#Replicating figure 7(b) from the paper
after = model_standard.data.moments_after
before = model_standard.data.moments_before

time = np.linspace(0, model.par.T_sim, model.par.T_sim)
plt.plot((time+1)*15, search_reference_beforeReform, color='Blue', label='Estima
plt.plot((time+1)*15, search_reference_afterReform, label='Estimated hazard, Aft
plt.plot((time+1)*15, before, label='Actual hazard, Before', color='Blue')
plt.plot((time+1)*15, after, label='Actual hazard, After', color='Red')
plt.xlabel('Days elapsed since UI claimed')
plt.ylabel('Hazard Rate')
# plt.title('Real and estimated hazard rates of the reference denpendence model'
plt.axvline(x=90, color='silver')
plt.axvline(x=270, color='silver')
plt.axvline(x=360, color='silver')
plt.legend()
plt.ylim(0.0, 0.08)
plt.show()
```



Model with **NO** reference dependence (standard model)

In [ ]:
```
# Setup
model_standard.par.eta = 0.0     # Removes reference dependence
model_standard.par.types = 3    # Allow for heterogeneity
model_standard.allocate()

# Estimation setup
est_par = ['delta','gamma', 'cost1', 'cost2', 'cost3', 'type_shares1', 'type_sha
theta0 = [0.9, 0.9, 84, 242, 310, 0.4, 0] # Initial guesses
bounds = [(0.5,1.0), (0.001, 50.0), (0.0, 100), (30, 300), (300, 1000), (0,0.9),

# Random starting values
model.par.noOfParams = len(est_par)
model.par.noSearchInits = 800
np.random.seed(180615)

est_best = np.inf

for i in range(model.par.noSearchInits):

    theta0 = [np.random.uniform(model.par.lb_delta, model.par.ub_delta),
              np.random.uniform(model.par.lb_gamma,model.par.ub_gamma),
              np.random.uniform(model.par.lb_lsc,model.par.ub_lsc),
              np.random.uniform(model.par.lb_msc,model.par.ub_msc),
              np.random.uniform(model.par.lb_hsc,model.par.ub_hsc),
              np.random.uniform(model.par.lb_share,model.par.ub_share),
              np.random.uniform(model.par.lb_share,model.par.ub_share)]
```

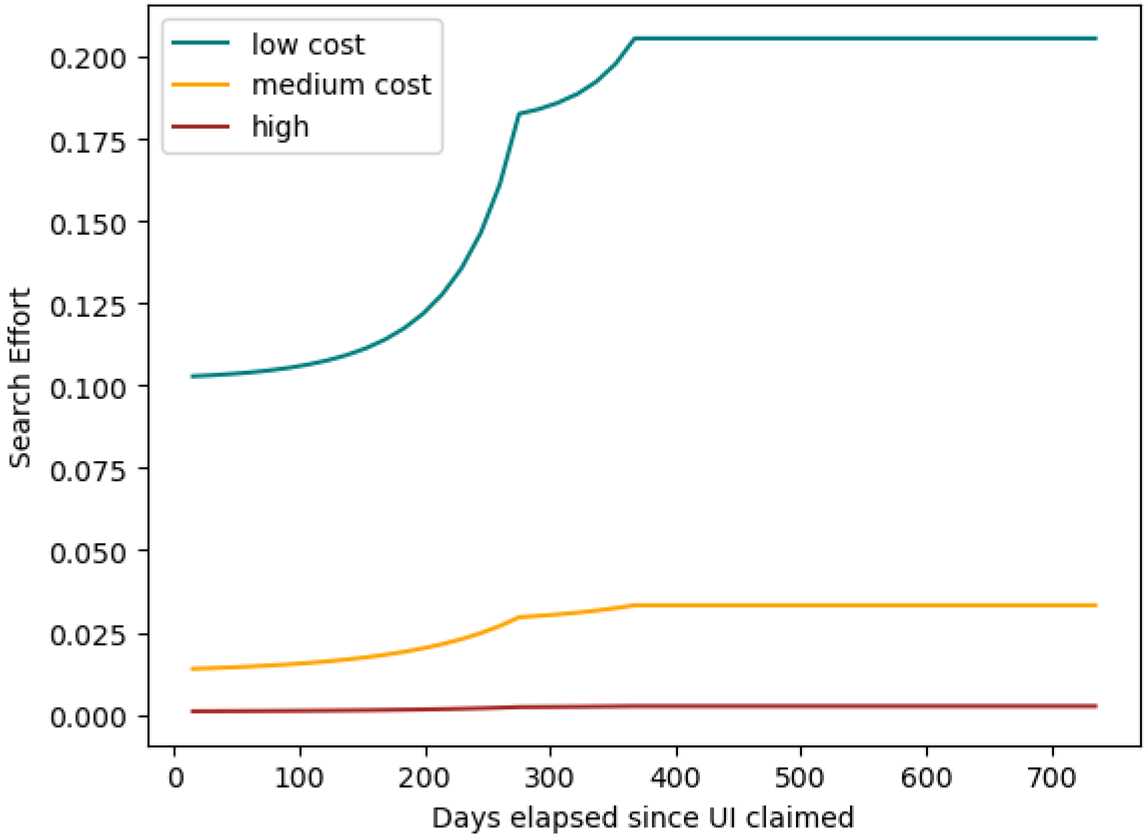
```
    try:
        # Perform the method of simulated moments
        est = method_simulated_moments(model_standard, est_par, theta0, bounds,

        # Check if the current estimate is better than the best estimate
        if est.fun < est_best:
            est_best = est.fun
            par = est.x
            nit = est.nit
            obj_standard = est_best

    except Exception as e:
        # Print the exception message and continue with the next iteration
        print(f"Iteration {i} failed with error: {e}")
        continue



print("Optimization Results:")
print("---------------------")
print(f'{"Parameter":<15} {"Estimate":<15}')
for param, estimate in zip(est_par, par):
    print(f'{param:<15} {estimate:>15.3f}')
print(f'Objective:                {est_best:.4f}')
print(f'Number of iterations:     {nit}')
```

```
d:\OneDrive\KU - Økonomi\Dynamic Programming\Term_Paper\Dynamic-programming-proje
ct\Funcs.py:65: RuntimeWarning: overflow encountered in scalar power
  inv_c_marg[1] = (s/par.cost2)**(1/par.gamma)
d:\OneDrive\KU - Økonomi\Dynamic Programming\Term_Paper\Dynamic-programming-proje
ct\Funcs.py:64: RuntimeWarning: overflow encountered in scalar power
  inv_c_marg[0] = (s/par.cost1)**(1/par.gamma)
d:\OneDrive\KU - Økonomi\Dynamic Programming\Term_Paper\Dynamic-programming-proje
ct\Funcs.py:64: RuntimeWarning: divide by zero encountered in scalar divide
  inv_c_marg[0] = (s/par.cost1)**(1/par.gamma)
d:\OneDrive\KU - Økonomi\Dynamic Programming\Term_Paper\Dynamic-programming-proje
ct\Funcs.py:45: RuntimeWarning: invalid value encountered in scalar multiply
  c[0] = par.cost1*s**(1+par.gamma)/(1+par.gamma)
d:\OneDrive\KU - Økonomi\Dynamic Programming\Term_Paper\Dynamic-programming-proje
ct\solve_hand_to_mouth.py:65: RuntimeWarning: invalid value encountered in scalar
multiply
  V_u[i,t] = utility(par,income,r) - cost(par,s[i,t])[i] + par.delta * (s[i,t] *
V_e_next+(1-s[i,t])*V_u[i,t+1])
Optimization Results:
---------------------
Parameter       Estimate
delta                     0.920
gamma                     0.331
cost1                    21.027
cost2                    64.656
cost3                   437.976
type_shares1              0.361
type_shares3              0.000
Objective:                0.4069
Number of iterations:     70
```

In [ ]:

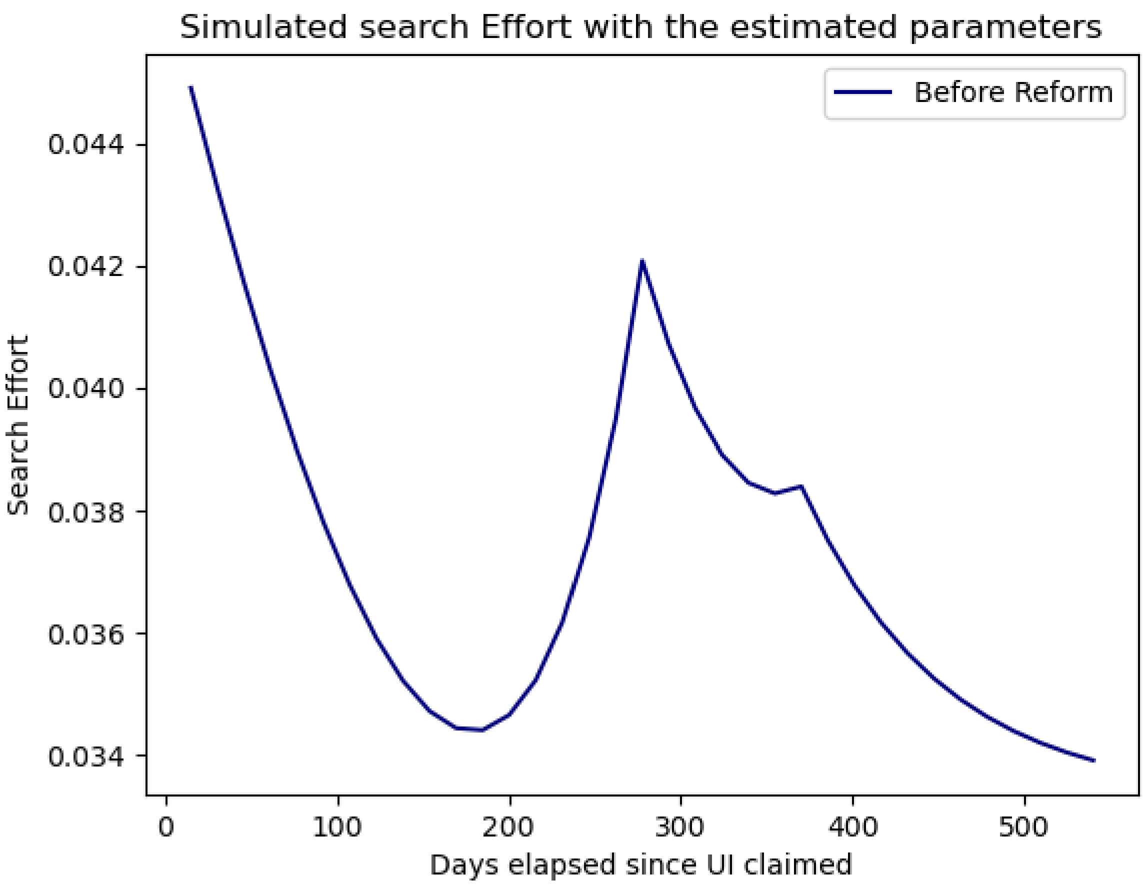
```
# Plot policy functions
search_effort_standard = solve_search_effort_HTM(model_standard.par)
time = np.linspace(0, model_standard.par.T, model_standard.par.T)
plt.plot((time+1)*15, search_effort_standard[0,:], label = 'low cost', color='te
```

```
plt.plot((time+1)*15, search_effort_standard[1,:], label = 'medium cost', color=
plt.plot((time+1)*15, search_effort_standard[2,:], label = 'high', color='brown'
plt.xlabel('Days elapsed since UI claimed')
plt.ylabel('Search Effort')
# plt.title('Policy functions standard model, After the reform')
plt.legend()
plt.show()
```

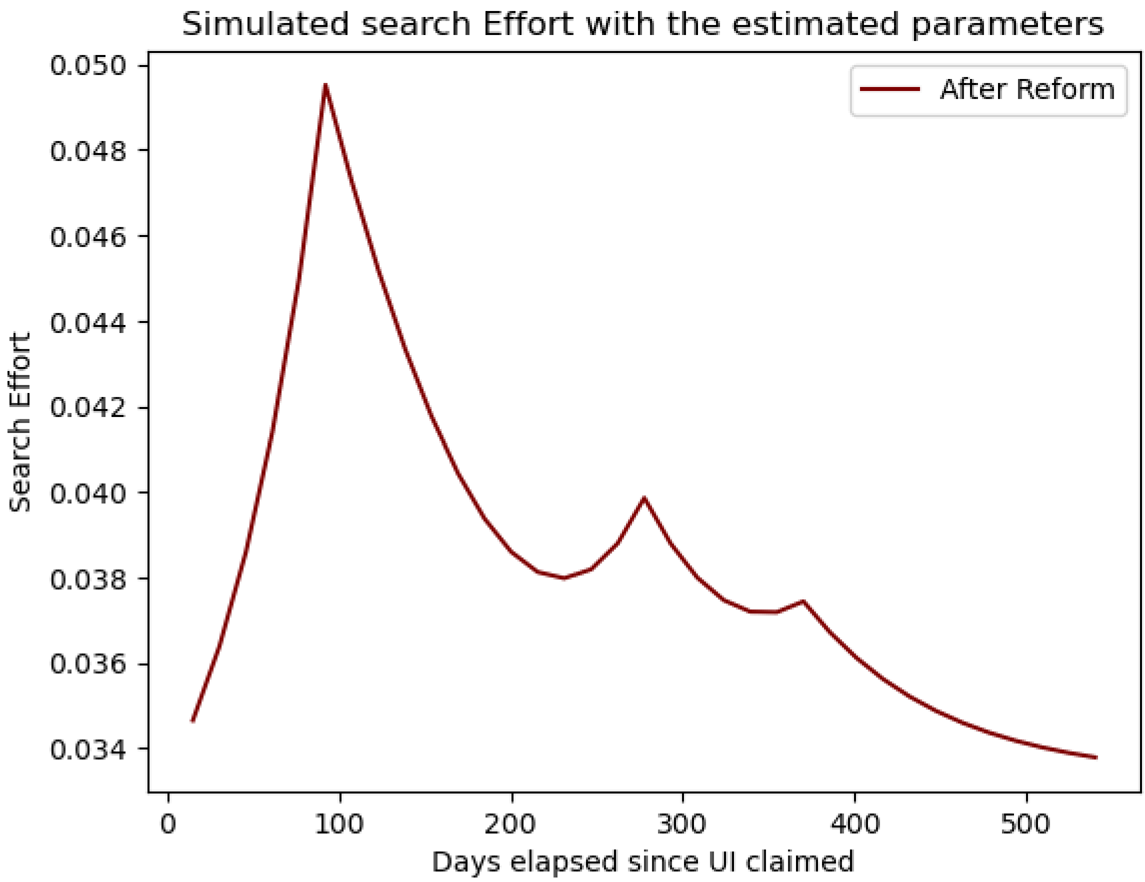


In [ ]:

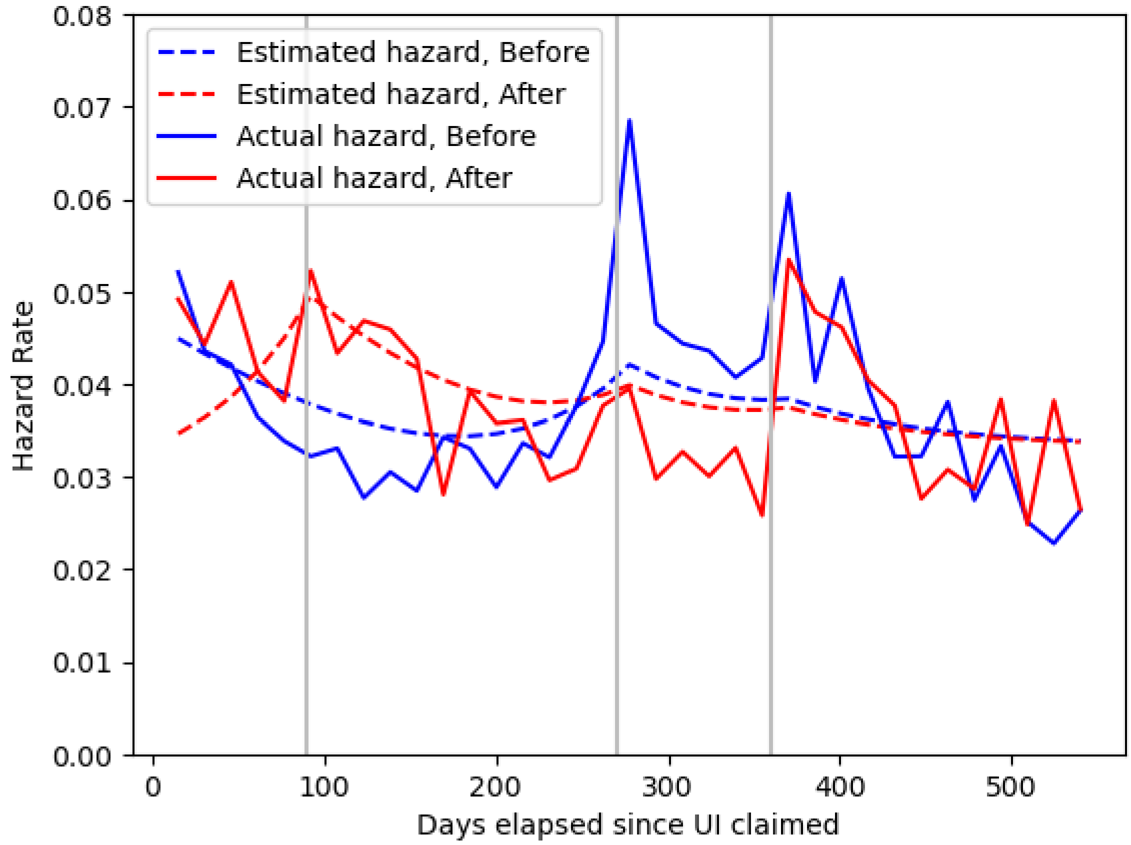
```
# Policy function before reform
model_standard.par.b1 = 222/675*model_standard.par.w    # High transfers
model_standard.par.b2 = model_standard.par.b1    # Medium transfers
model_standard.allocate()
search_standard_beforeReform = sim_search_effort_HTM(model_standard.par)
time = np.linspace(0, model_standard.par.T_sim, model_standard.par.T_sim)
plt.plot((time+1)*15, search_standard_beforeReform, label='Before Reform', color
plt.xlabel('Days elapsed since UI claimed')
plt.ylabel('Search Effort')
plt.title('Simulated search Effort with the estimated parameters')
plt.legend()
# plt.ylim(0.0, 0.08)
plt.show()
```



In [ ]:
```
# Policy function after reform
model_standard.par.b1 = 342.0/675.0      # Value after reform
model_standard.par.b2 = 171.0/675.0      # Value after reform

model_standard.allocate()
search_standard_afterReform = sim_search_effort_HTM(model_standard.par)


# Now plotting s_forecast
time = np.linspace(0, model_standard.par.T_sim, model_standard.par.T_sim)
plt.plot((time+1)*15, search_standard_afterReform, label='After Reform', color='
plt.xlabel('Days elapsed since UI claimed')
plt.ylabel('Search Effort')
plt.title('Simulated search Effort with the estimated parameters')
plt.legend()
# plt.ylim(0.0, 0.08)
plt.show()
```



In [ ]:
```
#Replicating figure 7(a) from the paper

time = np.linspace(0, model_standard.par.T_sim, model_standard.par.T_sim)
plt.plot((time+1)*15, search_standard_beforeReform, color='Blue', label='Estimat
plt.plot((time+1)*15, search_standard_afterReform, label='Estimated hazard, Afte
plt.plot((time+1)*15, before, label='Actual hazard, Before', color='Blue')
plt.plot((time+1)*15, after, label='Actual hazard, After', color='Red')
plt.xlabel('Days elapsed since UI claimed')
plt.ylabel('Hazard Rate')
# plt.title('Real and estimated hazard rates of the standard model')
plt.axvline(x=90, color='silver')
plt.axvline(x=270, color='silver')
plt.axvline(x=360, color='silver')
plt.legend()
plt.ylim(0.0, 0.08)
plt.show()
```



Comparison of the standard model and reference dependence

Through the Mean Square Error

In [ ]:
```
# true_data_afterReform = model_standard.data.moments_after
# true_data_beforeReform = model_standard.data.moments_before

# # Get the mean square errors
# mse_standard_afterReform = np.mean((true_data_afterReform - search_standard_af
# mse_standard_beforeReform = np.mean((true_data_beforeReform - search_standard_

# # Comparison after reform
# comparison1 = "smaller" if mse_reference_afterReform < mse_standard_afterRefor
# comparison2 = "WITH reference dependence" if mse_reference_afterReform < mse_s

# # Comparison before reform
# comparison3 = "smaller" if mse_reference_beforeReform < mse_standard_beforeRef
# comparison4 = "WITH reference dependence" if mse_reference_beforeReform < mse_


# print("Before the reform: ")
# print("-" * 100)
# print(f"The mean square error from the model \033[1mwith\033[0m reference depe
# print(f"The mean square error from the model \033[1mwithout\033[0m reference d
# print(f"The mean square error for the model \033[1mwith\033[0m reference depen
# print(f"Best model: \033[1m{comparison4}\033[0m.")
# print("-" * 100)
```

```
# print("After the reform:")
# print("-" * 100)
# print(f"The mean square error from the model \033[1mwith\033[0m reference depe
# print(f"The mean square error from the model \033[1mwithout\033[0m reference d
# print(f"The mean square error for the model \033[1mwith\033[0m reference depen
# print(f"Best model: \033[1m{comparison2}\033[0m.")
# print("-" * 100)
```

Through objective function

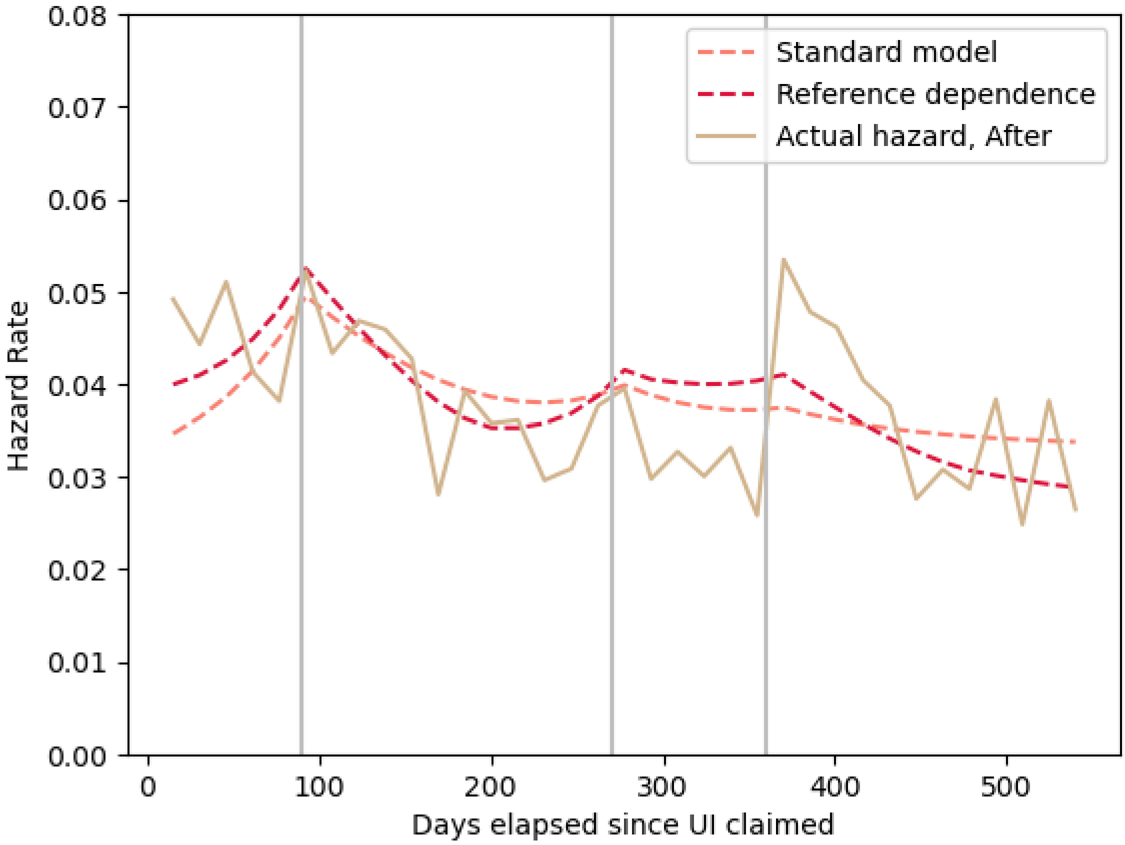
In [ ]:

```
# Comparison
comparison1 = "Reference Dependent Model" if obj_ref < obj_standard else "Standa


print("Comparison: ")
print("-" * 100)
print(f"The objective function from the model \033[1mwith\033[0m reference depen
print(f"The objective function from the model \033[1mwithout\033[0m reference de
print(f"Best model: \033[1m{comparison1}\033[0m.")
print("-" * 100)
```

```
Comparison:
--------------------------------------------------------------------------------
--------------------
The objective function from the model with reference dependence is 0.267209751457
61594
The objective function from the model without reference dependence is 0.406907198
794624
Best model: Reference Dependent Model.
--------------------------------------------------------------------------------
--------------------
```

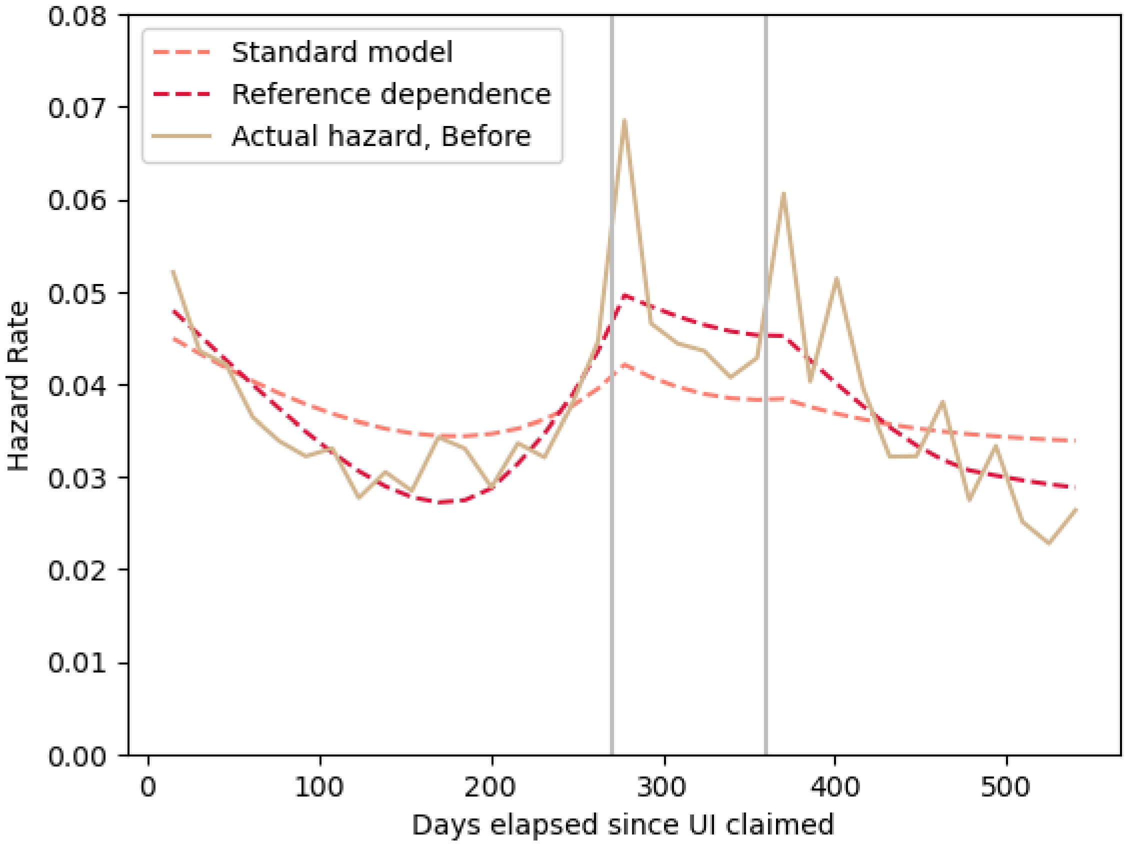
In [ ]:

```
after = model_standard.data.moments_after

time = np.linspace(0, model_standard.par.T_sim, model_standard.par.T_sim)     #
plt.plot((time+1)*15, search_standard_afterReform, label='Standard model', color
plt.plot((time+1)*15, search_reference_afterReform, label='Reference dependence'
plt.plot((time+1)*15, after, label='Actual hazard, After', color='tan')
# Make a vertical line at 90, 270 and 360
plt.axvline(x=90, color='silver')
plt.axvline(x=270, color='silver')
plt.axvline(x=360, color='silver')
plt.xlabel('Days elapsed since UI claimed')
plt.ylabel('Hazard Rate')
plt.legend()
# plt.title('Real and estimated hazard rates before the reform')
plt.ylim(0.0, 0.08)
plt.show()
```



In [ ]:

```
before = model_standard.data.moments_before

time = np.linspace(0, model_standard.par.T_sim, model_standard.par.T_sim)     #
plt.plot((time+1)*15, search_standard_beforeReform, label='Standard model', colo
plt.plot((time+1)*15, search_reference_beforeReform, label='Reference dependence
plt.plot((time+1)*15, before, label='Actual hazard, Before', color='tan')
plt.axvline(x=270, color='silver')
plt.axvline(x=360, color='silver')
plt.xlabel('Days elapsed since UI claimed')
plt.ylabel('Hazard Rate')
plt.legend()
# plt.title('Real and estimated hazard rates before the reform')
plt.ylim(0.0, 0.08)
plt.show()
```



Standard model
Reference dependence
Actual hazard, Before
Hazard Rate
Days elapsed since UI claimed
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0
100
200
300
400
500

In [ ]: