Yuval_Friedmann_ex9_computation_and_cognition

January 16, 2020

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id
=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire
ct_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20http
s%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.c
om%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.reado
nly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
ůůůůůůůůůů
Mounted at /content/drive
```

[0]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
os.chdir("/content/drive/My Drive/C&C/ex9_files")
from Hamster import myHamster
from scipy.io import loadmat
```

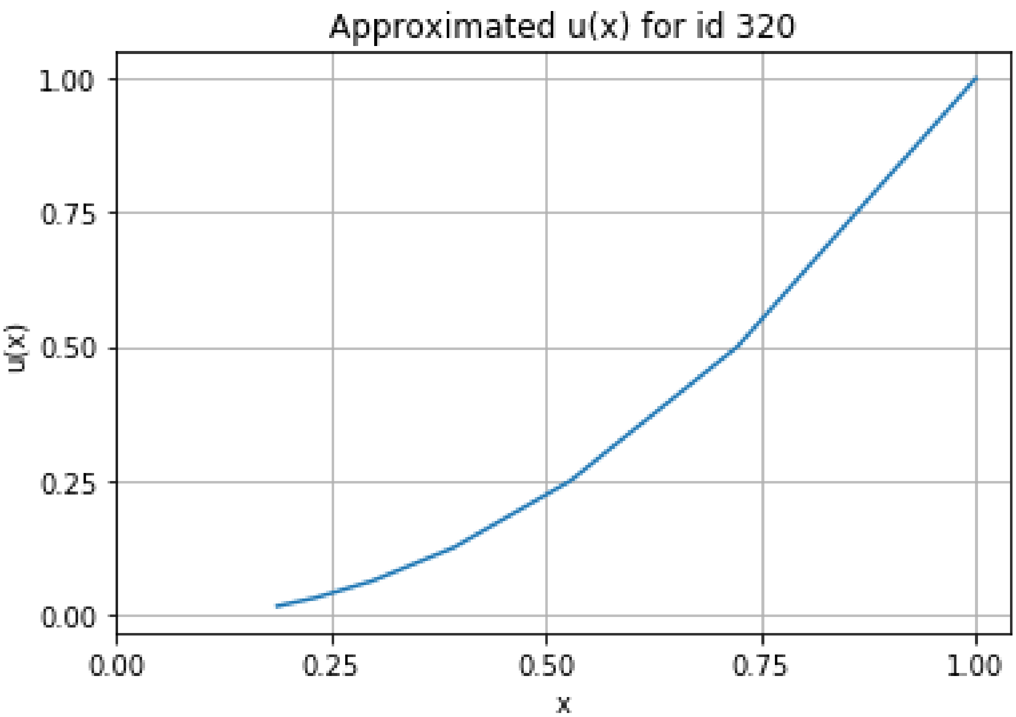
0.0.1 Q1

[0]:
```
# allows to implement many trails in efficient way
ham = np.vectorize(myHamster)
```

[0]:
```
trail=0
Xg = 1
bals = [1] # init Xs
u = [1] # init u(Xs)
while trail <=5:
  trail+=1
  balance = np.linspace(0,Xg,num=2000)[ham(np.linspace(0, Xg, num=2000),Xg,320).
↪sum()]
  bals.append(balance)
```

```
    u.append(u[trail-1]*0.5)
    Xg = balance
```

[6]:
```
plt.plot(bals,u)
plt.title("Approximated u(x) for id 320")
plt.xlabel("x")
plt.ylabel("u(x)")
plt.xticks(np.arange(0,1.25,0.25))
plt.yticks(np.arange(0,1.25,0.25))
plt.grid()
plt.show()
```



As we will explain later in Question 2.2, this shape of utility function represents "risk-loving", as the hamster under-appreciates the value of its possible gain.

0.0.2 Q2

[0]:
```
PI = "\u03C0"
ALPHA = "\u03B1"
SIGMA = "\u03C3"
```

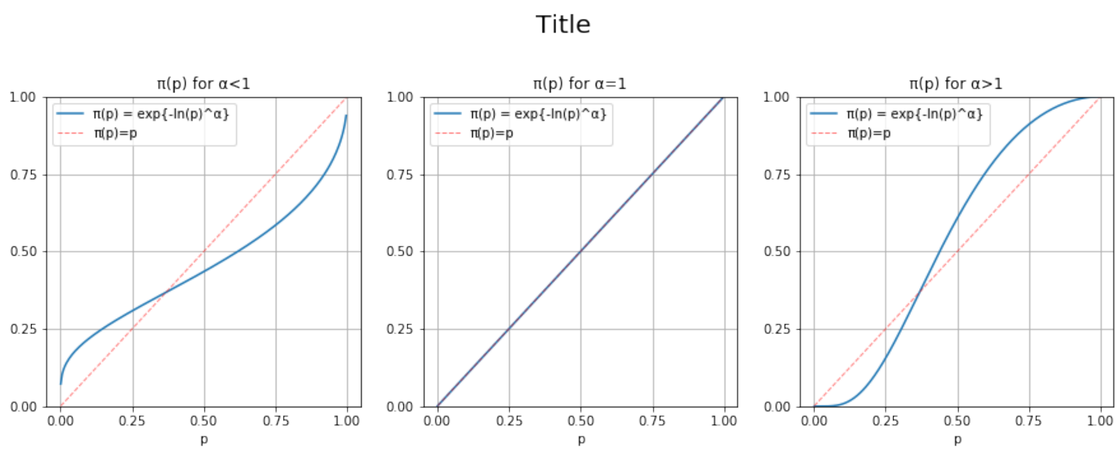
2.1

[8]:
```
p = np.arange(start=0.001,stop=1,step=0.005) # can't take log of zero
alpha = [0.5,1,1.9]
signs = ["<","=",">"]

fig, ax = plt.subplots(1,3, figsize=(15,5))
for i in range(len(alpha)):
  pi = np.exp(-((-np.log(p))**alpha[i]))

  ax[i].plot(p,pi, label=PI+"(p) = exp{-ln(p)^"+ALPHA+"}")
  ax[i].set_title(PI+"(p) for "+ALPHA+signs[i]+"1")
  ax[i].set_xlabel("p")
  ax[i].set_ylim(0,1)
  ax[i].grid()
  ax[i].plot(p,p,linestyle="--", alpha=0.5, lw=1, c="r", label=PI+"(p)=p")
  ax[i].set_xticks(np.arange(0,1.25,step=0.25))
  ax[i].set_yticks(np.arange(0,1.25,step=0.25))
  ax[i].legend()

fig.suptitle("Title", fontsize=20)
fig.subplots_adjust(top=0.8)
plt.show()
```



Differences between alpha's and the effect on the tendency to the gumbling option

- alpha<1:
  Overestimation of low probabilities and underestimation of high ones, corresponding with the "prospect theory". Suppose the hamster has an high p to get the peanuts in the gambling option, it would treat its decision as if it has lower probabilities to get the peanuts. On the other hand, it would be scared from the 1-p prob. to don't get the peanuts at all more than it should be. Therfore, it would choose the gambling option less times as it should be to maximize its gain expectation. In contrary, suppose the hamster has low p to get the peanuts, it would choose to gamble more times than it should be to maximize its gain expectation.

- alpha=1:
  The tendency of the hamster to gumble is determined directly by the real probabilities to gain the peanuts.

- alpha>1:
  Overestimation of high probabilities and underestimation of low ones. The reverse tendency of alpha<1: Suppuse to gumbling has high gain probabilities - the hamster would be even braver than it should be and would choose the this option a lot. Lower p to gain would cause to over fear from the hamster and it would tend not to gamble and choose the safe option even more than it should be.

Similarities between the alpha's The pi function "skew" the real probabilities, representing the probabilities as they are perceived in the hamster's mind (excluding alpha=1 exactly).

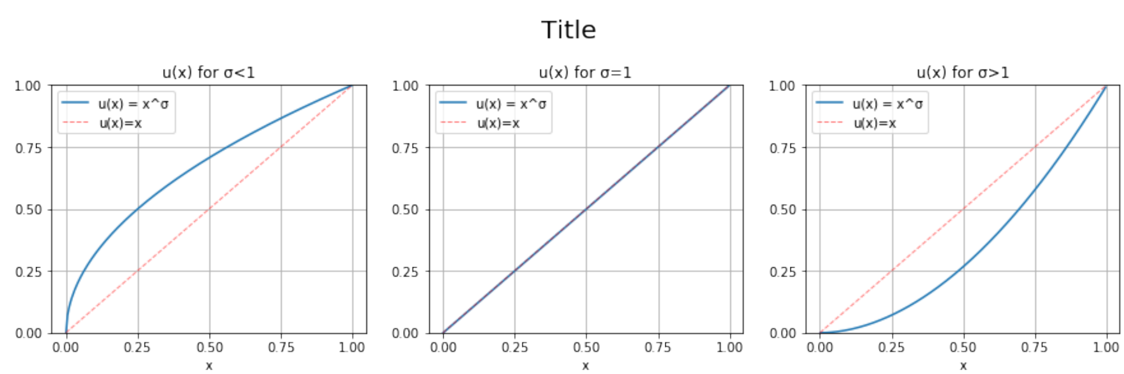
2.2

[9]:
```
x = np.arange(start=0,stop=1,step=0.005)
sigma = [0.5,1,1.9]
signs = ["<","=",">"]

fig, ax = plt.subplots(1,3, figsize=(15,4))
for i in range(len(sigma)):
  u = x**sigma[i]

  ax[i].plot(p,u, label="u(x) = x^"+SIGMA)
  ax[i].set_title("u(x) for "+SIGMA+signs[i]+"1")
  ax[i].set_xlabel("x")
  ax[i].set_ylim(0,1)
  ax[i].grid()
  ax[i].plot(x,x,linestyle="--", alpha=0.5, lw=1, c="r", label="u(x)=x")
  ax[i].set_xticks(np.arange(0,1.25,step=0.25))
  ax[i].set_yticks(np.arange(0,1.25,step=0.25))
  ax[i].legend()

fig.suptitle("Title", fontsize=20)
fig.subplots_adjust(top=0.8)
plt.show()
```



Similarities across the sigma's Mathematically, this function is a simple exponential function, and has its typical shapes for power by exponent which is smaller then (leftist plot) or greater then 1 (rightist plot). This function suppose to describe the subjective value of an x grams of peanuts as they are perceived in the hamster's mind. Naturally, the common thing across these three functions is that they are monotonically increase, namely, the more peanuts the hamster gets, the more he appreciates its gain.

Differences between sigma's and the effect on the tendency to the gumbling option

- sigma<1:
  Over appreciation of the value of gain by the hamster. In addition, notice that for lower amounts of peanuts the deviation of the value in the hamster's mind is remarkably higher then for higher ones (u(0.25)=0.5 but u(0.9) is almost equal 0.9. Typically in the safe option the amount of peanuts is lower then the gambling option, and as this case represent an over appreciation of peanuts, we could say that sigma<1 makes the subject to less selections of gambling. We named this situation "risk-aversion" in the class.

- sigma=1:
  The value of peanuts is the concrete amount of peanuts the hamster get.

- sigma>1:
  Under appreciation of the value of gain by the hamster. In addition, notice that for lower amounts of peanuts the deviation of the value in the hamster's mind is remarkably lower then for higher ones (u(0.25) is less then 0.1 but u(0.9) is almost equal 0.9). Typically in the safe option the amount of peanuts is lower then the gambling option, and as this case represent an under appreciation of peanuts, we could say that sigma>1 makes the subject to more selections of gambling. We named this situation "risk-loving" in the class.

2.3 - Analytic Part Attached in the end of the PDF

2.4

[0]:
```python
# load experiment from matlab
expr = pd.DataFrame()
for key in loadmat("ex9_q2_data.mat"):
  if type(loadmat("ex9_q2_data.mat")[key])==type(np.zeros((2,3))):
    expr = pd.concat([expr,pd.DataFrame(loadmat("ex9_q2_data.mat")[key].
 ↪reshape(-1),columns=[key])], axis=1)
expr["p"] = expr["p"].apply(lambda p: p/100)
expr["choice"] = expr["choice"].map({2:"s",1:"g"})
```

[0]:
```python
# Xg and p have one-to-one relationship, we'll drop p for make the processing␣
 ↪simpler
Xg_p_dict = {}
for t in expr[['Xg','p']].drop_duplicates().set_index("Xg").itertuples():
  Xg_p_dict[t[0]] = t[1]
```

[12]:
```
Xg_p_dict
```

[12]:
```
{150: 0.7, 400: 0.9, 500: 0.1, 700: 0.55, 2000: 0.06, 5000: 0.99, 10000: 0.002}
```

[0]:
```
# results with multindexing pandas
results = expr.groupby(["subject","h","Xg","choice"]).agg({"Xs":lambda x: x.
 ↪tolist()}).transpose()
```

[14]:
```
results
```

[14]:
```
subject        1   ...                                              30
h               1  ...                                               2
Xg          150    ...                                           10000
choice          g  ...                                               s
Xs       [50, 50]  ...  [3333, 6666, 1111, 2222, 370, 741, 122, 246]

[1 rows x 780 columns]
```

[0]:
```
''' rules according to the Note in the ex9 directions '''
def find_balance(row, Xg):
  if len(row.index) == 2:
    return np.mean([np.min(row["s"]), np.max(row["g"])])
  elif "s" in row.index:
    return np.mean([0,np.min(row["s"])])
  else: # only gambling
    return np.mean([Xg,np.max(row["g"])])


processing = pd.Series()
for i in range (1,31):
  for h in [1,2]:
    for Xg in Xg_p_dict.keys():
      balance = results[i,h,Xg].apply(lambda row: find_balance(row=row,Xg=Xg),␣
 ↪axis=1)
      processing.loc["_".join([str(i),str(h),str(Xg)])] = balance

processing = processing.apply(lambda x: x.values[0])
obs = pd.DataFrame.from_records(processing.index.str.split("_"),␣
 ↪columns=["subject","h","Xg"])
obs = obs.astype(int)

obs["p"] = obs["Xg"].apply(lambda Xg: Xg_p_dict[Xg])
obs["Xs"] = processing.values
# depended and independed variables as we found in q. 2.3
obs["x"] = np.log(-np.log(obs["p"]))
obs["y"] = np.log(-np.log(obs["Xs"]/obs["Xg"]))
```

[0]:
```
lm = obs.groupby(["subject","h"])["x","y"].agg(lambda x: x.tolist())
params = lm.apply(lambda row: np.polyfit(row["x"], row["y"], 1), axis=1)
alpha = params.apply(lambda lst: lst[0]) # unpacking alpha
```

```python
sigma = params.apply(lambda lst: np.exp(-lst[1])) # unpacking sigma
lm["alpha"], lm["sigma"] = alpha , sigma
```

[17]:
```python
lm[["alpha", "sigma"]]
```

[17]:

subject	h	alpha	sigma
1	1	0.378442	0.464979
	2	0.610302	0.564987
2	1	0.589914	0.578437
	2	0.618548	0.597233
3	1	0.682697	0.887746
	2	0.645998	0.794806
4	1	0.384650	0.838725
	2	0.314972	0.898790
5	1	0.620445	0.532713
	2	0.522244	0.495497
6	1	0.389769	0.918626
	2	0.533485	0.690198
7	1	0.456037	0.483448
	2	0.457004	0.456831
8	1	0.676004	0.800892
	2	0.711994	0.733048
9	1	0.769450	0.934794
	2	0.611956	0.619133
10	1	0.543197	0.491510
	2	0.535405	0.541161
11	1	0.411423	0.709683
	2	0.432206	0.584678
12	1	0.575588	0.872567
	2	0.584147	0.839775
13	1	1.033464	0.625075
	2	0.766778	0.540716
14	1	0.380338	0.679702
	2	0.289411	0.619241
15	1	0.293541	0.375520
	2	0.444094	0.589892
16	1	-0.072533	1.980592
	2	-0.009272	2.644401
17	1	0.407244	0.447173
	2	0.596016	0.502608
18	1	0.414652	0.734284
	2	0.330619	0.783610
19	1	1.042908	0.682559
	2	1.119167	0.941372
20	1	0.648744	0.632246
	2	1.021031	0.699661
21	1	0.617958	0.647357

```
         2  0.535320  0.754299
22       1  0.414125  0.735159
         2  0.444371  0.733055
23       1  0.425893  0.811819
         2  0.436144  0.907007
24       1  0.477762  0.360133
         2  0.464239  0.387211
25       1  1.045792  0.541137
         2  1.163953  0.712723
26       1  0.504150  0.864473
         2  0.437273  0.891141
27       1  0.599635  0.632968
         2  0.507010  0.645594
28       1  0.434432  0.395500
         2  0.468394  0.423263
29       1  0.598530  0.651507
         2  0.680775  0.558440
30       1  0.618067  0.764151
         2  0.451280  0.632176
```

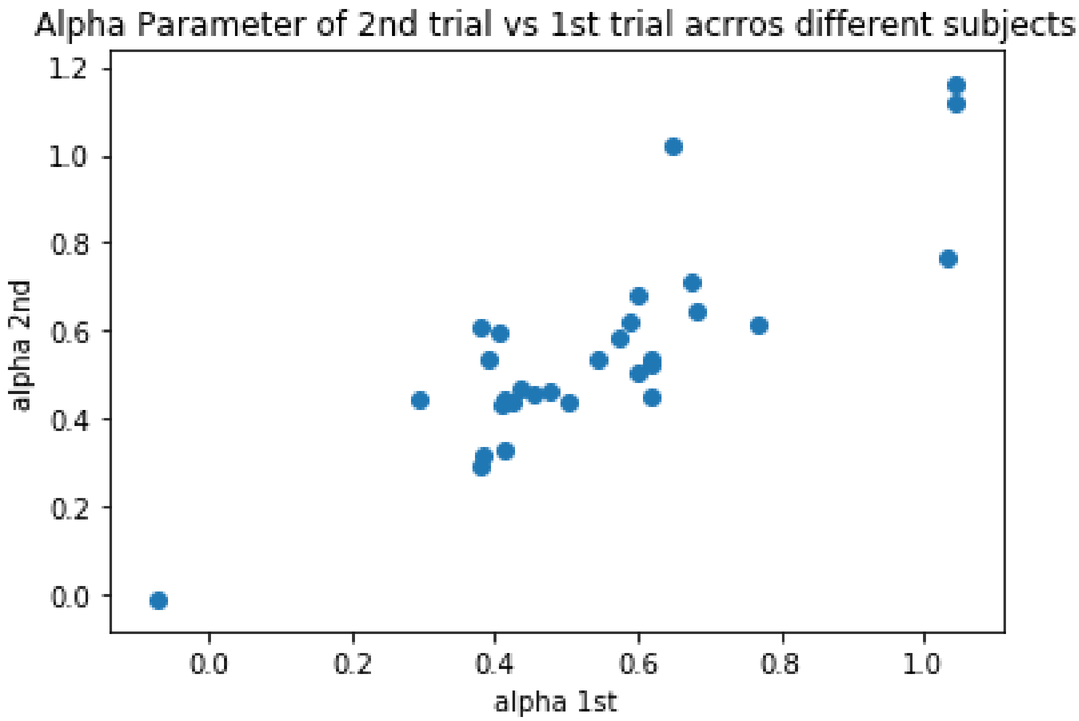
[18]:
```
# we've got few negative alpha's but we have to
# remember we didn't constrain our regression to positivies
# and it's possible that the OLS linear estimator will deviate a bit
# from the real estimator, as far as this values are closed to zero
lm[["alpha", "sigma"]].agg(["min","max"])
```

[18]:
```
        alpha     sigma
min -0.072533  0.360133
max  1.163953  2.644401
```

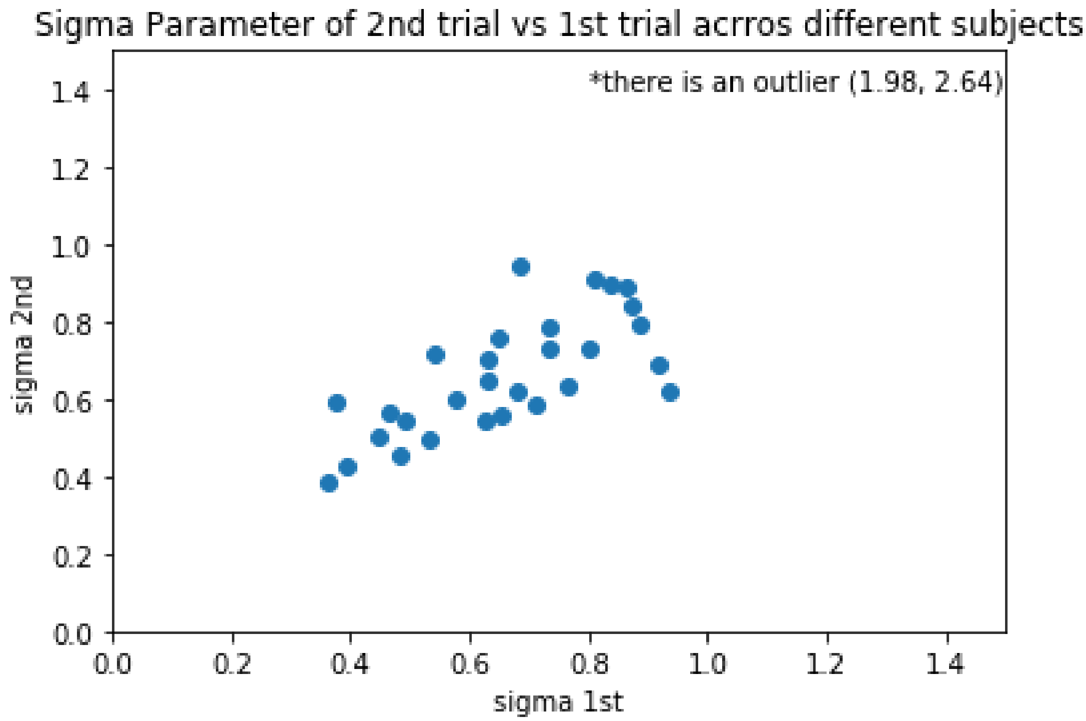
2.5

[0]:
```
to_plot = lm[["alpha", "sigma"]].groupby("h").agg(lambda x: x.tolist())
```

[20]:
```
plt.scatter(to_plot.loc[1,"alpha"], to_plot.loc[2,"alpha"])
plt.title("Alpha Parameter of 2nd trial vs 1st trial acrros different subjects")
plt.xlabel("alpha 1st")
plt.ylabel("alpha 2nd")
plt.show()
```



[21]:
```
plt.scatter(to_plot.loc[1,"sigma"], to_plot.loc[2,"sigma"])
plt.title("Sigma Parameter of 2nd trial vs 1st trial acrros different subjects")
plt.text(0.8,1.4, s="*there is an outlier (1.98, 2.64)")
plt.xlabel("sigma 1st")
plt.xlim(0,1.5)
plt.ylim(0,1.5)
plt.ylabel("sigma 2nd")
plt.show()
```



Broadly speaking, it seems each subject preserves his estimated parameters alpha and sigma in the two trials. It's interesting to see that for most of the subjects sigma is bewteen 0 to 1, which implies that they are "risk-averted". In addition, the fact that for most of the subjects alpha is bewteen 0 to 1 is corresponded to the "Prospect-Theory" of kahneman and Tversky.

Average alpha and sigma across all subjects and values of h:

[22]:
```
lm.mean()
```

[22]:
```
alpha    0.551453
sigma    0.714300
dtype: float64
```

2.3

<u>חישוביות עצבית תרגיל 9</u>

יוסי פרידמן 204169320

לפי א׳ ההגדרה מתקיים

$$E[\text{gain} \mid \text{safe}] = E[\text{gain} \mid \text{gambling}]$$

$$\Rightarrow \overbrace{u(x_s)}^{\text{הרווח כאן דטרמיניסטי}} = \pi(p)\cdot u(x_g) + \underbrace{\pi(1-p)\cdot u(0)}_{0}$$

$$\Rightarrow x_s^{\sigma} = e^{-(-\ln p)^{\alpha}}\cdot x_g^{\sigma} + 0 \quad / \ln(\cdot)$$

$$\Rightarrow \sigma \ln(x_s) = -(-\ln p)^{\alpha} + \sigma \ln(x_g)$$

$$\Rightarrow \ln(x_s) - \ln(x_g) = -\frac{(-\ln p)^{\alpha}}{\sigma}$$

$$\Rightarrow -\ln\left(\frac{x_s}{x_g}\right) = \frac{(-\ln p)^{\alpha}}{\sigma} \quad / \ln(\cdot)$$

$$\Rightarrow \ln\left(-\ln\left(\frac{x_s}{x_g}\right)\right) = \alpha \ln(-\ln(p)) - \ln(\sigma)$$