Table of contents

In [ ]:
```
import numpy as np
from types import SimpleNamespace
```

1. Problem 1: Production economy and CO2 taxation

Consider a production economy with two firms indexed by $j \in \{1,2\}$. Each produce its own good. They solve

$$\begin{aligned}\max_{y_j} \pi_j &= p_j y_j - w_j \ell_j \\ \text{s.t. } y_j &= A\ell_j^{\gamma}.\end{aligned}$$

Optimal firm behavior is

$$\begin{aligned}\ell_j^{\star}(w,p_j) &= \left(\frac{p_j A\gamma}{w}\right)^{\frac{1}{1-\gamma}} \\ y_j^{\star}(w,p_j) &= A\left(\ell_j^{\star}(w,p_j)\right)^{\gamma}\end{aligned}$$

The implied profits are

$$\pi_j^*(w,p_j) = \frac{1-\gamma}{\gamma}w\cdot\left(\frac{p_j A\gamma}{w}\right)^{\frac{1}{1-\gamma}}$$

A single consumer supplies labor, and consumes the goods the firms produce. She also recieves the implied profits of the firm.
She solves:

$$\begin{aligned}U(p_1,p_2,w,\tau,T) = \max_{c_1,c_2,\ell} &\log(c_1^{\alpha}c_2^{1-\alpha}) - \nu\frac{\ell^{1+\epsilon}}{1+\epsilon} \\ \text{s.t. } &p_1c_1 + (p_2+\tau)c_2 = w\ell + T + \pi_1^*(w,p_1) + \pi_2^*(w,p_2)\end{aligned}$$

where τ is a tax and T is lump-sum transfer.
For a given ℓ, it can be shown that optimal behavior is

$$\begin{aligned}c_1(\ell) &= \alpha\frac{w\ell + T + \pi_1^*(w,p_1) + \pi_2^*(w,p_2)}{p_1} \\ c_2(\ell) &= (1-\alpha)\frac{w\ell + T + \pi_1^*(w,p_1) + \pi_2^*(w,p_2)}{p_2+\tau}\end{aligned}$$

Such that optimal behavior is:

$$\ell^* = \arg\max_{\ell} \log((c_1(\ell))^{\alpha} \cdot (c_2(\ell))^{1-\alpha}) - \nu\frac{\ell^{1+\epsilon}}{1+\epsilon}$$

With optimal consumption:

$$c_1^* = c_1(\ell^*)$$
$$c_2^* = c_2(\ell^*)$$

The government chooses τ and balances its budget so $T = \tau c_2^*$. We initially set $\tau, T = 0$.

Market clearing requires:

1. Labor market: $\ell^* = \ell_1^* + \ell_2^*$
2. Good market 1: $c_1^* = y_1^*$
3. Good market 2: $c_2^* = y_2^*$

Question 1: Check market clearing conditions for p_1 in `linspace(0.1,2.0,10)` and p_2 in `linspace(0.1,2.0,10)`. We choose $w = 1$ as numeraire.

In [ ]:
```
par = SimpleNamespace()

# firms
par.A = 1.0
par.gamma = 0.5

# households
par.alpha = 0.3
par.nu = 1.0
par.epsilon = 2.0

# government
par.tau = 0.0
par.T = 0.0

# Question 3
par.kappa = 0.1
```

In [ ]:
```
%load_ext autoreload
%autoreload 2
from Problem_1 import CO2Model
```

In [ ]:
```
CO2 = CO2Model(par)
CO2.solve_grid()
```

```
p1_ast = 0.94, p2_ast = 1.58
```

Question 2: Find the equilibrium prices p_1 and p_2.

In [ ]:
```
CO2.solve(do_print=True)
```

```
sol.p1 = 0.98, sol.p2 = 1.49
errors = (4.278454812656207e-10, 6.628760873539363e-10, 9.765484430701576e-05)
```

Assume the government care about the social welfare function:

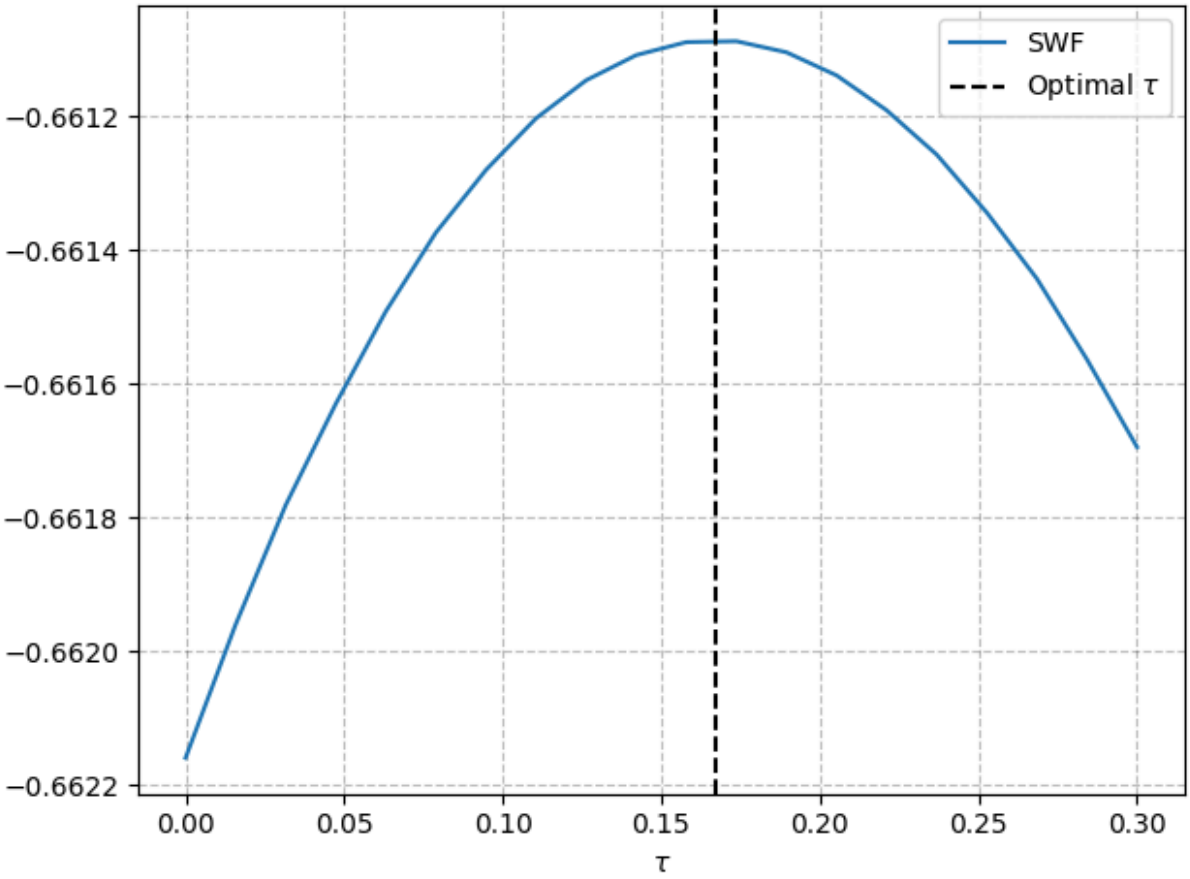
$$SWF = U - \kappa y_2^*$$

Here κ measures the social cost of carbon emitted by the production of y_2 in equilibrium.

Question 3: What values of τ and (implied) T should the government choose to maximize SWF?

In [ ]:
```
CO2.optimal_gov()
```

```
Grid search:
tau = 0.000, sol.U = -0.5876, sol.y2 = 0.7454, SWF = -0.66216
tau = 0.016, sol.U = -0.5876, sol.y2 = 0.7433, SWF = -0.66196
tau = 0.032, sol.U = -0.5877, sol.y2 = 0.7412, SWF = -0.66178
tau = 0.047, sol.U = -0.5877, sol.y2 = 0.7391, SWF = -0.66163
tau = 0.063, sol.U = -0.5878, sol.y2 = 0.7370, SWF = -0.66149
tau = 0.079, sol.U = -0.5879, sol.y2 = 0.7350, SWF = -0.66137
tau = 0.095, sol.U = -0.5880, sol.y2 = 0.7329, SWF = -0.66128
tau = 0.111, sol.U = -0.5881, sol.y2 = 0.7309, SWF = -0.66120
tau = 0.126, sol.U = -0.5883, sol.y2 = 0.7288, SWF = -0.66115
tau = 0.142, sol.U = -0.5884, sol.y2 = 0.7268, SWF = -0.66111
tau = 0.158, sol.U = -0.5886, sol.y2 = 0.7248, SWF = -0.66109
tau = 0.174, sol.U = -0.5888, sol.y2 = 0.7228, SWF = -0.66109
tau = 0.189, sol.U = -0.5890, sol.y2 = 0.7208, SWF = -0.66110
tau = 0.205, sol.U = -0.5893, sol.y2 = 0.7188, SWF = -0.66114
tau = 0.221, sol.U = -0.5895, sol.y2 = 0.7169, SWF = -0.66119
tau = 0.237, sol.U = -0.5898, sol.y2 = 0.7149, SWF = -0.66126
tau = 0.253, sol.U = -0.5900, sol.y2 = 0.7130, SWF = -0.66134
tau = 0.268, sol.U = -0.5903, sol.y2 = 0.7111, SWF = -0.66144
tau = 0.284, sol.U = -0.5907, sol.y2 = 0.7091, SWF = -0.66156
tau = 0.300, sol.U = -0.5910, sol.y2 = 0.7072, SWF = -0.66169

Optimal tau:
tau = 0.167, sol.U = -0.5887, sol.y2 = 0.7237, SWF = -0.66109
Gvining optimal T = 0.12
```



2. Problem 2: Career choice model

Consider a graduate i making a choice between entering J different career tracks. Entering career j yields utility u_{ij}^k. This value is unknown to the graduate ex ante, but will ex post be:

$$u_{i,j}^k = v_j + \epsilon_{i,j}^k$$

They know that $\epsilon_{i,j}^k \sim \mathcal{N}(0, \sigma^2)$, but they do not observe $\epsilon_{i,j}^k$ before making their career choice.

Consider the concrete case of $J = 3$ with:

$$v_1 = 1$$
$$v_2 = 2$$
$$v_3 = 3$$

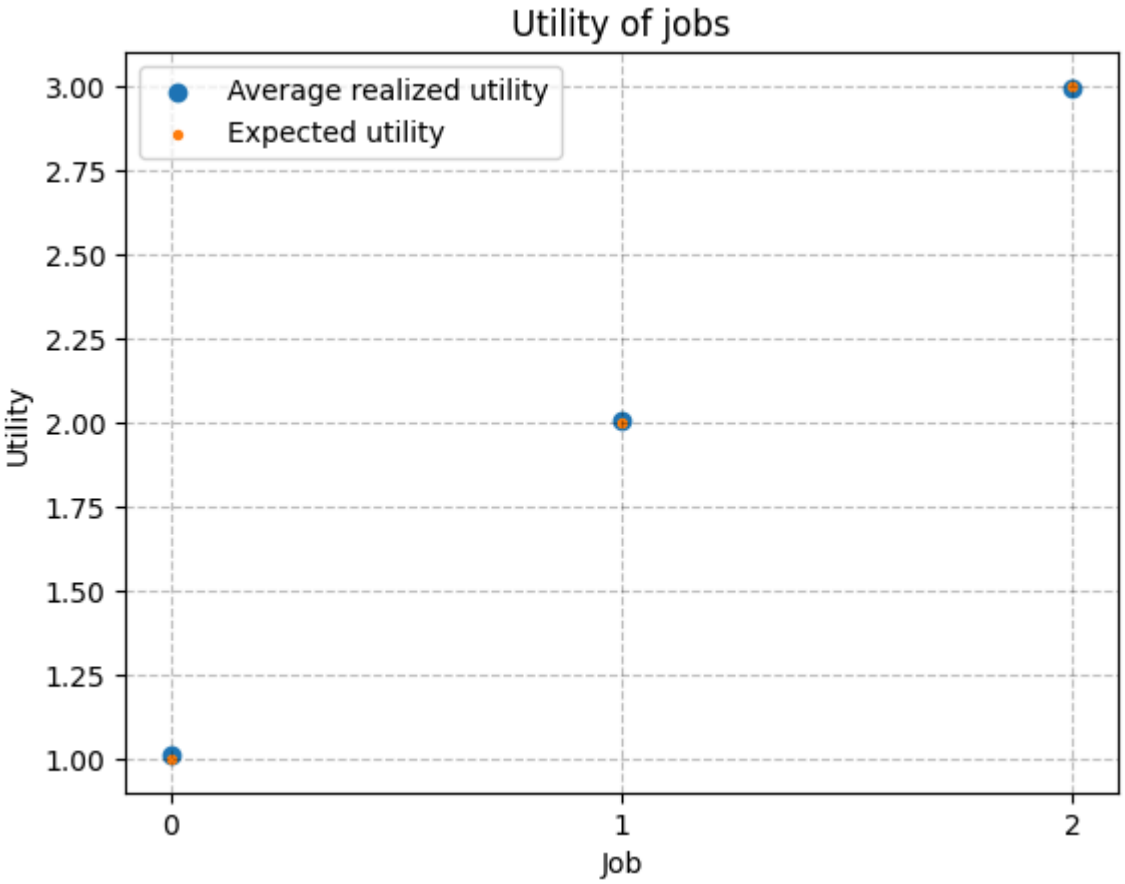
If the graduates know the values of v_j and the distribution of $\epsilon_{i,j}^k$, they can calculate the expected utility of each career track using simulation:

$$\mathbb{E}\left[u_{i,j}^k|v_j\right] \approx v_j + \frac{1}{K}\sum_{k=1}^{K}\epsilon_{i,j}^k$$

In [ ]:
```
par = SimpleNamespace()
par.J = 3
par.N = 10
par.K = 10000

par.F = np.arange(1,par.N+1)
par.sigma = 2

par.v = np.array([1,2,3])
par.c = 1
```

Question 1: Simulate and calculate expected utility and the average realised utility for $K = 10000$ draws, for each career choice j.

In [ ]:
```
from Problem_2 import CareerChoiceModel
morty = CareerChoiceModel(**par.__dict__)
sim = morty.sim
par = morty.par

np.random.seed(123)
morty.simulate()
morty.plot_post_utility()
```



Now consider a new scenario: Imagine that the graduate does not know v_j. The *only* prior information they have on the value of each job, comes from their F_i friends that work in each career j. After talking with them, they know the average utility of their friends (which includes their friends' noise term), giving them the prior expecation:

$$\tilde{u}^k_{i,j}\left(F_i\right) = \frac{1}{F_i}\sum_{f=1}^{F_i}\left(v_j + \epsilon^k_{f,j}\right),\ \epsilon^k_{f,j} \sim \mathcal{N}(0,\sigma^2)$$

For ease of notation consider that each graduate have $F_i = i$ friends in each career.

For K times do the following:

1. For each person i draw $J \cdot F_i$ values of $\epsilon^k_{f,j}$, and calculate the prior expected utility of each career track, $\tilde{u}^k_{i,j}\left(F_i\right)$.
   Also draw their own J noise terms, $\epsilon^k_{i,j}$
2. Each person i chooses the career track with the highest expected utility:

$$j^{k*}_i = \arg\max_{j \in 1,2\ldots,J}\left\{\tilde{u}^k_{i,j}\left(F_i\right)\right\}$$

3. Store the chosen careers: j^{k*}_i, the prior expectation of the value of their chosen career: $\tilde{u}^k_{i,j=j^{k*}_i}\left(F_i\right)$, and the realized value of their chosen career track: $u^k_{i,j=j^{k*}_i} = v_{j=j^{k*}_i} + \epsilon^k_{i,j=j^{k*}_i}$.

Chosen values will be:
$i \in \{1, 2 \ldots, N\}, N = 10$
$F_i = i$
So there are 10 graduates. The first has 1 friend in each career, the second has 2 friends, ... the tenth has 10 friends.

Question 2: Simulate and visualize: For each type of graduate, i, the share of graduates choosing each career, the average subjective expected utility of the graduates, and the average ex post realized utility given their career choice.
That is, calculate and visualize:

$$\frac{1}{K} \sum_{k=1}^{K} \mathbb{I}\left\{j = j_i^{k*}\right\} \ \forall j \in \{1, 2, \ldots, J\}$$

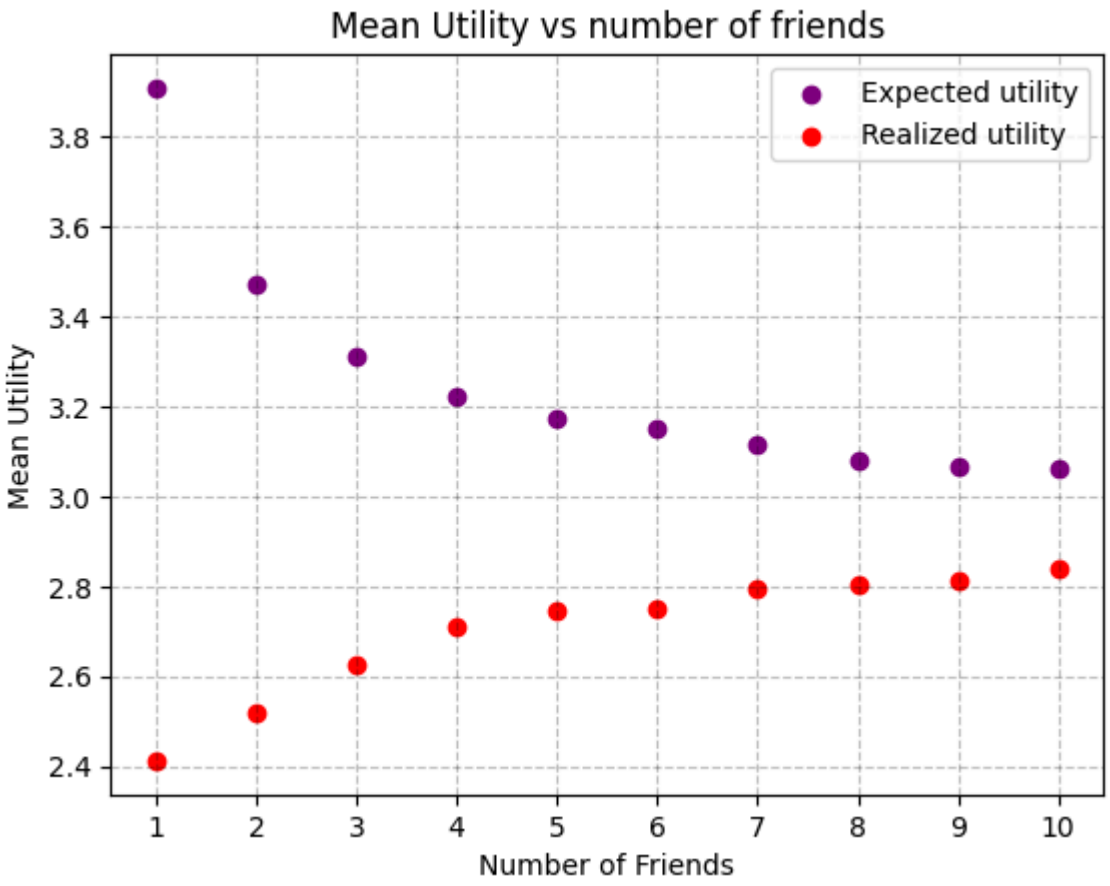
$$\frac{1}{K} \sum_{k=1}^{K} \tilde{u}^k_{ij=j_i^{k*}}(F_i)$$

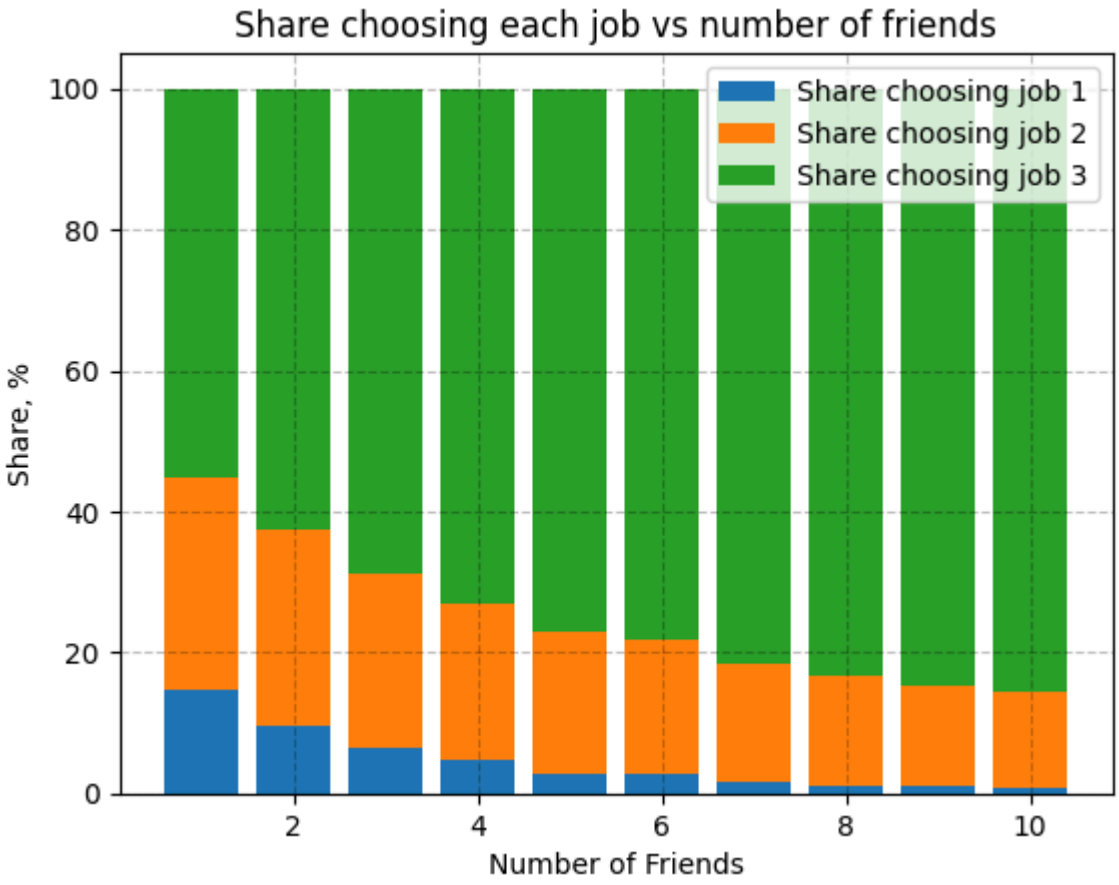
And

$$\frac{1}{K} \sum_{k=1}^{K} u^k_{ij=j_i^{k*}}$$

For each graduate i.

In [ ]:
```
morty.utility()
morty.plot_utility()
```


Mean Utility vs number of friends
Expected utility
Realized utility
Mean Utility
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
1
2
3
4
5
6
7
8
9
10
Number of Friends


Share choosing each job vs number of friends
Share choosing job 1
Share choosing job 2
Share choosing job 3
Share, %
100
80
60
40
20
0
2
4
6
8
10
Number of Friends

After a year of working in their career, the graduates learn u_{ij}^{k} for their chosen job j_i^{k*} perfectly.
The can switch to one of the two remaining careers, for which they have the same prior as before, but it will now include a switching cost of c which is known. Their new priors can be written as:

$$\tilde{u}_{ij}^{k,2}\left(F_i\right) = \begin{cases} \tilde{u}_{ij}^{k}\left(F_i\right) - c & \text{if } j \neq j_i^{k*} \\ u_{ij=j_i^{k*}} & \text{if } j = j_i^{k*} \end{cases}$$

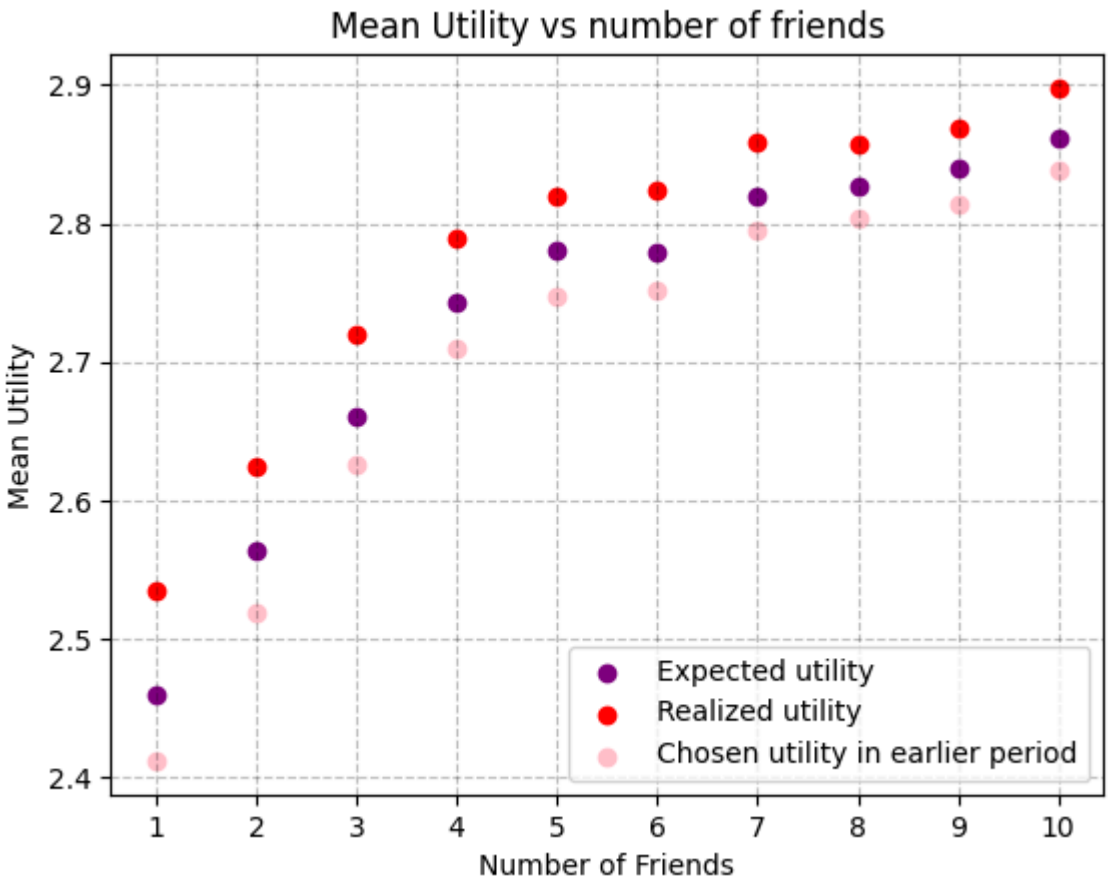
We will set $c = 1$.

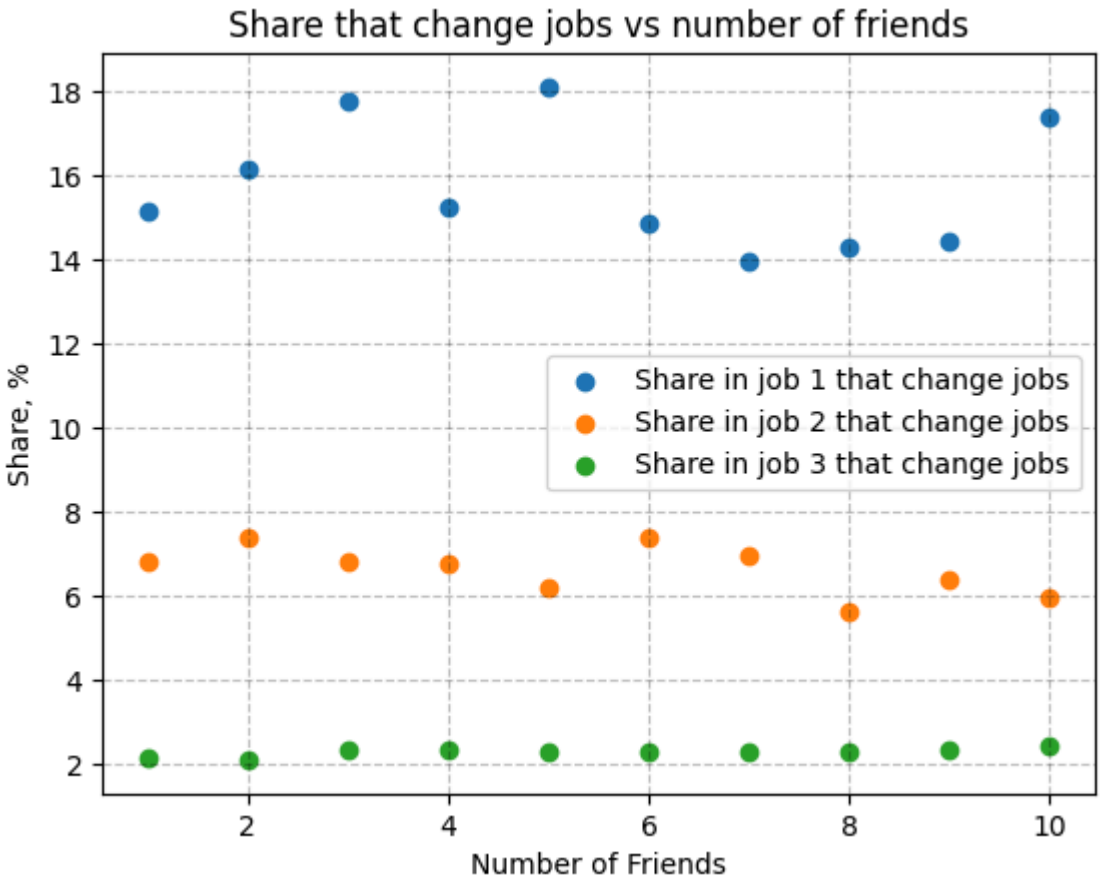
Their realized utility will be:

$$u_{ij}^{k,2} = \begin{cases} u_{ij}^{k} - c & \text{if } j \neq j_i^{k*} \\ u_{ij=j_i^{k*}} & \text{if } j = j_i^{k*} \end{cases}$$

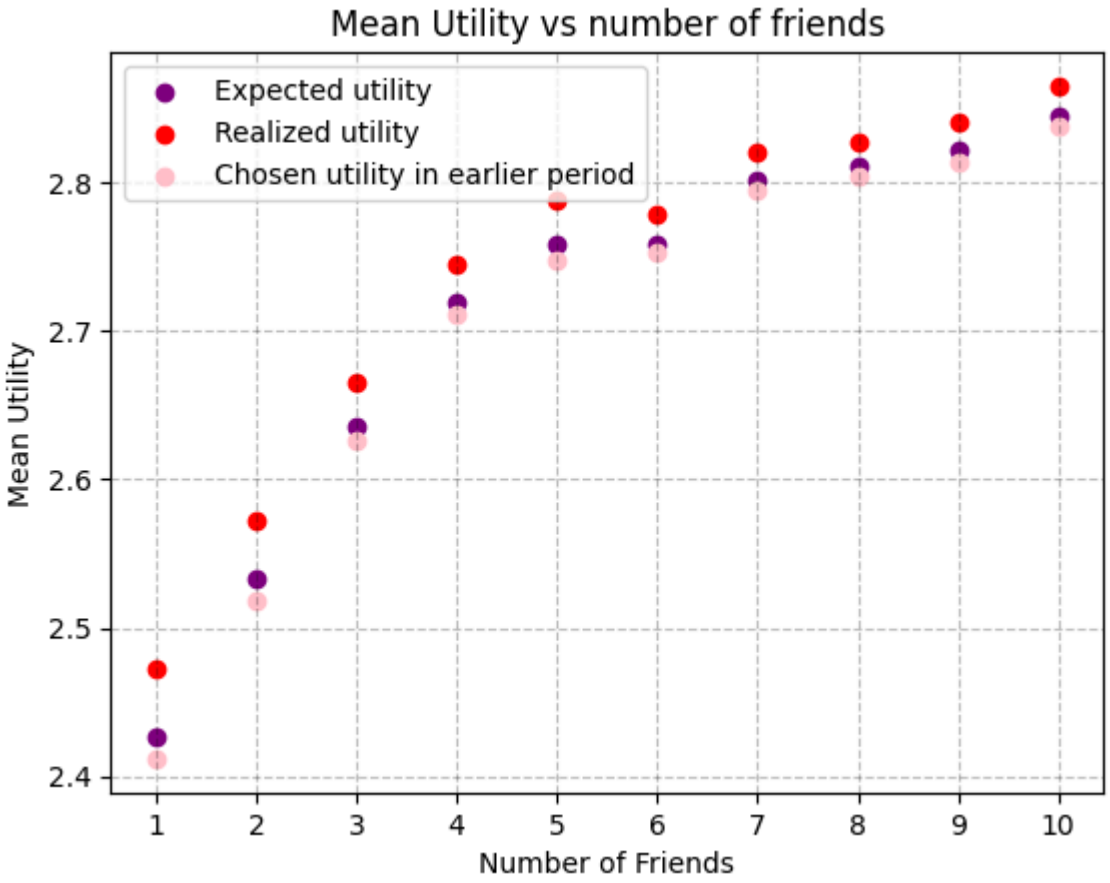
Question 3: Following the same approach as in question 2, find the new optimal career choice for each i, k. Then for each i, calculate the average subjective expected utility from their new optimal career choice, and the ex post realized utility of that career. Also, for each i , calculate the share of graduates that chooses to switch careers, conditional on which career they chose in the first year.

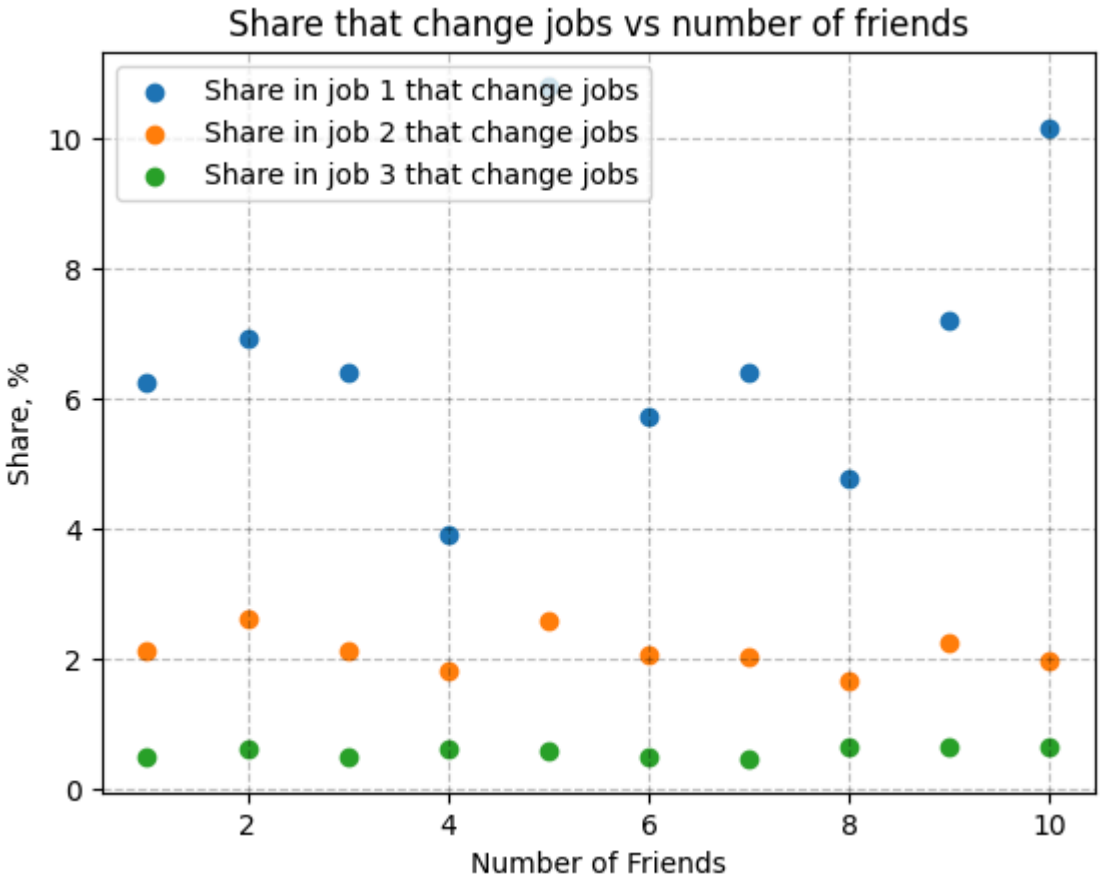
In [ ]:
```
morty.career_change()
morty.plot_utility_change()
```


Mean Utility vs number of friends
Mean Utility
Number of Friends
Expected utility
Realized utility
Chosen utility in earlier period
2.9
2.8
2.7
2.6
2.5
2.4
1
2
3
4
5
6
7
8
9
10


Share that change jobs vs number of friends
Share, %
Number of Friends
Share in job 1 that change jobs
Share in job 2 that change jobs
Share in job 3 that change jobs
18
16
14
12
10
8
6
4
2
2
4
6
8
10

In [ ]:
```
morty.career_change()
morty.plot_utility_change()
```





3. Problem 3: Barycentric interpolation

Problem: We have a set of random points in the unit square,

$$\mathcal{X} = \{(x_1, x_2) \,|\, x_1 \sim \mathcal{U}(0,1), x_2 \sim \mathcal{U}(0,1)\}.$$

For these points, we know the value of some function $f(x_1, x_2)$,

$$\mathcal{F} = \{f(x_1, x_2) \,|\, (x_1, x_2) \in \mathcal{X}\}.$$

Now we want to approximate the value $f(y_1, y_2)$ for some $y = (y_1, y_2)$, where $y_1 \sim \mathcal{U}(0,1)$ and $y_2 \sim \mathcal{U}(0,1)$.

Building block I

For an arbitrary triangle ABC and a point y, define the so-called barycentric coordinates as:

$$\begin{aligned} r_1^{ABC} &= \frac{(B_2 - C_2)(y_1 - C_1) + (C_1 - B_1)(y_2 - C_2)}{(B_2 - C_2)(A_1 - C_1) + (C_1 - B_1)(A_2 - C_2)} \\ r_2^{ABC} &= \frac{(C_2 - A_2)(y_1 - C_1) + (A_1 - C_1)(y_2 - C_2)}{(B_2 - C_2)(A_1 - C_1) + (C_1 - B_1)(A_2 - C_2)} \\ r_3^{ABC} &= 1 - r_1 - r_2. \end{aligned}$$

If $r_1^{ABC} \in [0,1]$, $r_2^{ABC} \in [0,1]$, and $r_3^{ABC} \in [0,1]$, then the point is inside the triangle.

We always have $y = r_1^{ABC} A + r_2^{ABC} B + r_3^{ABC} C$.

Building block II

Define the following points:

$$A = \arg \min_{(x_1,x_2)\in\mathcal{X}} \sqrt{(x_1-y_1)^2+(x_2-y_2)^2} \text{ s.t. } x_1 > y_1 \text{ and } x_2 > y_2$$

$$B = \arg \min_{(x_1,x_2)\in\mathcal{X}} \sqrt{(x_1-y_1)^2+(x_2-y_2)^2} \text{ s.t. } x_1 > y_1 \text{ and } x_2 < y_2$$

$$C = \arg \min_{(x_1,x_2)\in\mathcal{X}} \sqrt{(x_1-y_1)^2+(x_2-y_2)^2} \text{ s.t. } x_1 < y_1 \text{ and } x_2 < y_2$$

$$D = \arg \min_{(x_1,x_2)\in\mathcal{X}} \sqrt{(x_1-y_1)^2+(x_2-y_2)^2} \text{ s.t. } x_1 < y_1 \text{ and } x_2 > y_2.$$

Algorithm:

1. Compute A, B, C, and D. If not possible return `NaN`.
2. If y is inside the triangle ABC return $r_1^{ABC} f(A) + r_2^{ABC} f(B) + r_3^{ABC} f(C)$.
3. If y is inside the triangle CDA return $r_1^{CDA} f(C) + r_2^{CDA} f(D) + r_3^{CDA} f(A)$.
4. Return `NaN`.

Sample:

In [ ]:
```
rng = np.random.default_rng(2024)

X = rng.uniform(size=(50,2))
y = rng.uniform(size=(2,))
```

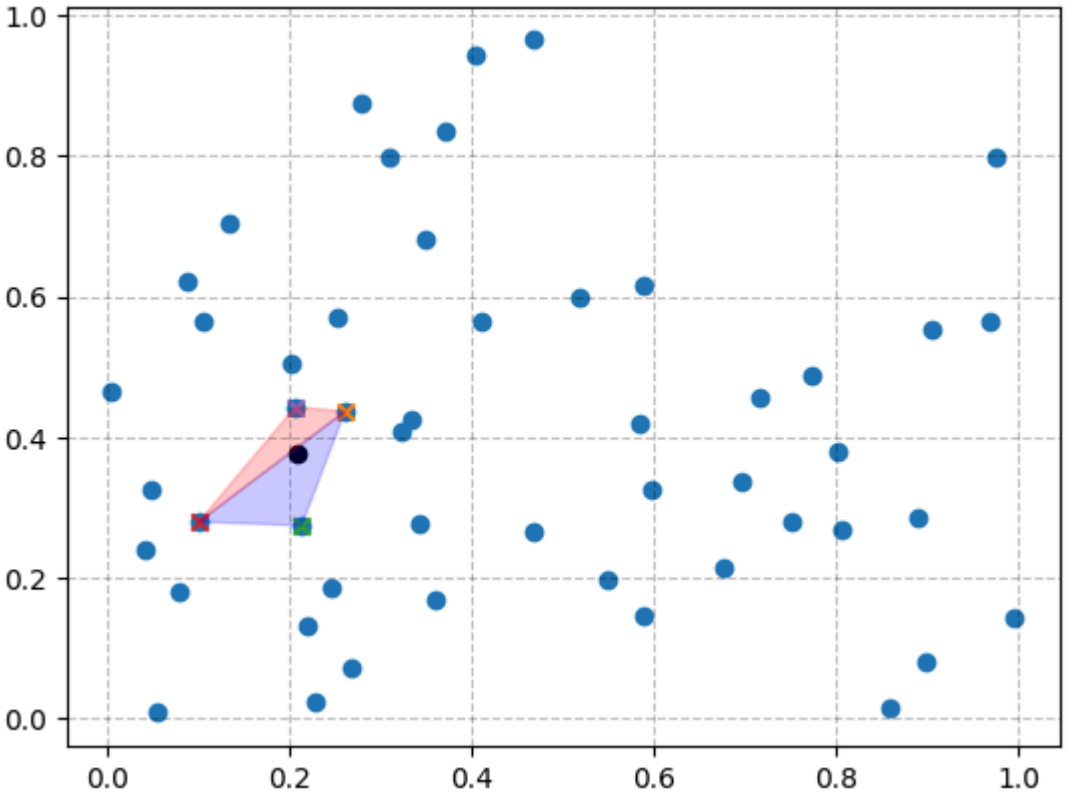
Questions 1: Find A, B, C and D. Illustrate these together with X, y and the triangles ABC and CDA.

In [ ]:
```
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
plt.rcParams.update({"axes.grid":True,"grid.color":"black","grid.alpha":"0.25","gri
```

In [ ]:
```
def find_ABCD(X,y):

    # upper right
    I = (X[:,0] > y[0]) & (X[:,1] > y[1])
    i = np.argmin(np.sqrt((X[I,0]-y[0])**2+(X[I,1]-y[1])**2))
    A = np.array([X[I,0][i],X[I,1][i]])

    # lower right
    I = (X[:,0] > y[0]) & (X[:,1] < y[1])
    i = np.argmin(np.sqrt((X[I,0]-y[0])**2+(X[I,1]-y[1])**2))
    B = np.array([X[I,0][i],X[I,1][i]])

    # lower left
    I = (X[:,0] < y[0]) & (X[:,1] < y[1])
```

```
    i = np.argmin(np.sqrt((X[I,0]-y[0])**2+(X[I,1]-y[1])**2))
    C = np.array([X[I,0][i],X[I,1][i]])

    # upper left
    I = (X[:,0] < y[0]) & (X[:,1] > y[1])
    i = np.argmin(np.sqrt((X[I,0]-y[0])**2+(X[I,1]-y[1])**2))
    D = np.array([X[I,0][i],X[I,1][i]])

    return A,B,C,D
```

In [ ]:
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)

ax.scatter(X[:,0],X[:,1])
ax.scatter(y[0],y[1],color='black')

A,B,C,D = find_ABCD(X,y)

ax.scatter(A[0],A[1],marker='x')
ax.scatter(B[0],B[1],marker='x')
ax.scatter(C[0],C[1],marker='x')
ax.scatter(D[0],D[1],marker='x')

ax.add_patch(Polygon([A,B,C],closed=True,color='blue',alpha=0.2))
ax.add_patch(Polygon([C,D,A],closed=True,color='red',alpha=0.2));
```



Question 2: Compute the barycentric coordinates of the point y with respect to the triangles ABC and CDA. Which triangle is y located inside?

In [ ]:
```
def find_barycentric_coordinates(x,A,B,C):

    denom = (B[1]-C[1])*(A[0]-C[0]) + (C[0]-B[0])*(A[1]-C[1])
    r1 = ((B[1]-C[1])*(x[0]-C[0]) + (C[0]-B[0])*(x[1]-C[1]))/denom
    r2 = ((C[1]-A[1])*(x[0]-C[0]) + (A[0]-C[0])*(x[1]-C[1]))/denom
    r3 = 1 - r1 - r2

    inside = (r1 >= 0) & (r2 >= 0) & (r3 >= 0) & (r1 <= 1) & (r2 <= 1) & (r3 <= 1)

    return np.array([r1,r2,r3]),inside
```

In [ ]:
```
r_ABC,inside_ABC = find_barycentric_coordinates(y,A,B,C)
r_CDA,inside_CDA = find_barycentric_coordinates(y,C,D,A)

assert np.allclose(y,r_ABC[0]*A + r_ABC[1]*B + r_ABC[2]*C)
assert np.allclose(y,r_CDA[0]*C + r_CDA[1]*D + r_CDA[2]*A)

print('ABC',r_ABC,inside_ABC)
print('CDA',r_CDA,inside_CDA)
```

```
ABC [0.62862632 0.06910145 0.30227223] True
CDA [ 0.36825988 -0.13392662  0.76566674] False
```

Now consider the function:

$$f(x_1, x_2) = x_1 \cdot x_2$$

In [ ]:
```
f = lambda x: x[0]*x[1]
F = np.array([f(x) for x in X])
```

Question 3: Compute the approximation of $f(y)$ using the full algorithm. Compare with the true value.

In [ ]:
```
fy_approx = r_ABC[0]*f(A) + r_ABC[1]*f(B) + r_ABC[2]*f(C)
print(f'{f(y) = : .4f}, {fy_approx = : .4f}')
```

```
f(y) =  0.0790, fy_approx =  0.0841
```

Question 4: Repeat question 3 for all points in the set Y.

In [ ]:
```
Y = [(0.2,0.2),(0.8,0.2),(0.8,0.8),(0.8,0.2),(0.5,0.5)]
```

In [ ]:
```
fY_true = np.array([f(y_) for y_ in Y])
fY_approx = np.zeros(fY_true.shape)

for i,y_ in enumerate(Y):

    try:

        A,B,C,D = find_ABCD(X,y_)

    except:

        fY_approx[i] = np.nan
```

```
        continue

    r_ABC,inside_ABC = find_barycentric_coordinates(y_,A,B,C)

    if inside_ABC:

        fY_approx[i] = r_ABC[0]*f(A) + r_ABC[1]*f(B) + r_ABC[2]*f(C)
        continue

    r_CDA,inside_CDA = find_barycentric_coordinates(y_,C,D,A)
    if inside_CDA:

        fY_approx[i] = r_CDA[0]*f(C) + r_CDA[1]*f(D) + r_CDA[2]*f(A)
        continue

    fY_approx[i] = np.nan

for i,y_ in enumerate(Y):
    print(f'{y_[0] = : .4f}, {y_[1] = : .4f}: {fY_true[i] = : .4f}, {fY_approx[i] =
```

```
y_[0] =  0.2000, y_[1] =  0.2000: fY_true[i] =  0.0400, fY_approx[i] =  0.0403
y_[0] =  0.8000, y_[1] =  0.2000: fY_true[i] =  0.1600, fY_approx[i] =  0.1587
y_[0] =  0.8000, y_[1] =  0.8000: fY_true[i] =  0.6400, fY_approx[i] =  nan
y_[0] =  0.8000, y_[1] =  0.2000: fY_true[i] =  0.1600, fY_approx[i] =  0.1587
y_[0] =  0.5000, y_[1] =  0.5000: fY_true[i] =  0.2500, fY_approx[i] =  0.2513
```