

## 5. Generel ligevaegt

Skriv dit eksamensnummer her: 45

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as wg
%matplotlib inline
```

I en model er der tre agenter. De har Cobb-Douglas præferencer med nyttefunktion:

$$U_i(c_{i,1}, c_{i,2}, c_{i,3}, f_i) = c_{i,1}^{1/3} c_{i,2}^{1/3} c_{i,3}^{1/3} f_i,$$

hvor  $i = 1, 2, 3$ ,  $c_{i,j}$  er agent  $i$ 's forbrug af gode  $j$  og  $f_i \in (0, 1)$  er fritid.

Der er tre industrier, alle under perfekt konkurrence. De har produktionsteknologi:

$$y_i = L_i^{2/3},$$

hvor  $L_i = 1 - f_i$  er arbejdstid og  $\log$  er den naturlige logaritme. Agent  $i$  er den eneste der kan arbejde i industri  $i$ .

### Opgave 5.a)

- Løs forbruger  $i$ 's problem for givne priser  $(p_1, p_2, p_3)$ , løn  $w_i$  og profit  $\pi_i$  analytisk (dvs. find optimalt  $c_{i,j}$  og  $f_i$ )
- Skriv løsningen ind i nedenstående funktion, der tager priser, løn og profit som input og returnerer optimalt forbrugsvalg og fritid for forbruger  $i$

```
In [2]: def løsning_forbrugeren(p, w, pi):
'''Løs forbrugers problem for forbruger i

Parametre
-----
p: NumPy array med 3 elementer: (p1, p2, p3)
w: Løn for forbruger i
pi: Profit for virksomhed i

Returnerer
-----
c: NumPy array med 3 elementer: Optimalt (c1, c2, c3)
f: Optimal fritid
...
c = []
for i in range(0,p.size):
    c.append(1/2* 1/sum(p) * (w+pi) / p[i])
f = 0.5 * (w+pi) / w

return c, f
```

Hint: Hvis du eksekverer følgende celle, skal den printe `c = [0.142 0.177 0.118]` og `f = 0.5`

```
In [3]: p = np.array([1, 0.8, 1.2])
w = 0.85
pi = 0.0
c, f = løsning_forbrugeren(p, w, pi)
print(f'x = {np.round(c,3)}')
print(f'f = {np.round(f,3)}')

x = [0.142 0.177 0.118]
f = 0.5
```

## Opgave 5.b)

- Løs virksomhed  $i$ 's problem for givne  $(p_i, w_i)$  analytisk (dvs. find optimalt  $L_i$ )
- Skriv løsningen ind i nedenstående funktion, der tager pris og løn som input og returnerer optimal arbejdstid, produktion og profit for virksomhed  $i$

```
In [4]: def løsning_virksomheden(p, w):
'''Løs virksomhedens problem for virksomhed i

Parametre
-----
p: Pris på vare i
w: Løn for agent i

Returnerer
-----
L: Optimal arbejdstid efterspurgt af virksomhed i
y: Optimal produktion af vare i
pi: Optimal profit for virksomhed i
...

L = (8 * p**3) / (27 * w**3)
y = L**(2/3)
pi = (4 * p**3) / (27 * w**2)

return L, y, pi
```

*Hint:* Hvis du eksekverer følgende celle, skal den printe `L = 0.097`, `y = 0.211` og `pi = 0.07`.

```
In [5]: p = 1
w = 1.45
L, y, pi = løsning_virksomheden(p, w)
print(f'L = {np.round(L,3)}')
print(f'y = {np.round(y,3)}')
print(f'pi = {np.round(pi,3)}')

L = 0.097
y = 0.211
pi = 0.07
```

## Opgave 5.c)

- Beskriv hvad de to funktioner i følgende celler gør

*Bemærk: Opgaven fortsætter under cellen*

Funktionen `optimering` finder de optimale værdier ( $L, y, pi, c, f$ ) ud fra en given pris- og

lønvektor ved brug af funktionerne ovenfor, (løsning\_virkksomheden(p, w) og løsning\_forbrugeren(p, w, pi)), der giver den analytiske løsning på henholdsvis virksomheden og forbrugers problem.

Herefter benytter funktionen residualer\_markedsclearing(p, w) de optimale løsninger til at bestemme residualerne i markederne

```
In [6]: def optimering(p, w):
    ...

    Parametre
    -----
    p: NumPy Array med 3 elemeter: (p1, p2, p3)
    w: NumPy Array med 3 elementer: (w1, w2, w3)

    Returnerer
    -----
    L: NumPy Array med 3 elemeter: (L1, L2, L3)
    y: NumPy Array med 3 elementer: (y1, y2, y3)
    pi: NumPy Array med 3 elementer: (pi1, pi2, pi3)
    c: NumPy Array med (3,3) elementer: Element (i,j) er agent i's forbrug af vare
    f: NumPy Array med 3 elementer: (f1, f2, f3)

    ...

    L = np.full(3, np.nan)
    y = np.full(3, np.nan)
    pi = np.full(3, np.nan)
    for i in range(3):
        L[i], y[i], pi[i] = løsning_virkksomheden(p[i], w[i])

    c = np.full((3,3), np.nan)
    f = np.full(3, np.nan)
    for i in range(3):
        c[i,:], f[i] = løsning_forbrugeren(p, w[i], pi[i])

    return L, y, pi, c, f

def residualer_markedsclearing(p, w):
    ...

    Parametre
    -----
    p: NumPy Array med 3 elemeter: (p1, p2, p3)
    w: NumPy Array med 3 elementer: (w1, w2, w3)

    ...

    L, y, pi, c, f = optimering(p, w)

    residual_varemarked = np.sum(c, axis=0) - y
    residual_arbejdsmarked = L - (1-f)

    res_alle = np.concatenate([residual_varemarked, residual_arbejdsmarked])

    return res_alle
```

Sæt prisen på vare 1 som numeraire, altså  $p_1 = 1$ . Bemærk at alle forbrugere og virksomheder er identiske og at alle forbrugsgoder indgår identisk i nyttefunktionen. I

løsningen gælder der defor, at  $p_1 = p_2 = p_3 = 1$  og  $w_1 = w_2 = w_3 = w^*$ . Den fælles løn ( $w^*$ ) er endnu ukendt.

- Lav en funktion, der tager  $w^*$  som argument (input). Funktionen skal plotte residualerne på de tre varemarkeder og de tre arbejdsmarkeder i et barplot ( `plt.bar` ). y-aksens grænser skal være  $-3$  og  $3$ . *Hint: Benyt  $p_1 = p_2 = p_3 = 1$  og  $w_1 = w_2 = w_3 = w^*$  og `residualer_markedsclearing` \*.*
- Lav en floatslider for  $w^*$  fra 0.5 til 2 med skridt af 0.01, der styrer figuren. For (ca.) hvilken værdi af  $w^*$  clearer alle 6 markeder?

```
In [7]: # SKRIV DIN LØSNING HER:
p = np.array([1,1,1])
w = 1

def resPlot(w):
    wage = np.array(3 * [w])
    res = residualer_markedsclearing(p,wage)
    for i in range(0,res.size):
        s = 'residual' + str(i+1)
        h = [-3,3]
        plt.bar(s, res[i])
        plt.ylim(-3,3)

w_float_slider = wg.FloatSlider(
    value = 0.5,
    min=0.5,
    max=2,
    step = 0.01,
    description = '$w$',
    continuous_update = True)
wg.interact(resPlot, w=w_float_slider)

interactive(children=(FloatSlider(value=0.5, description='$w$', max=2.0, min=0.5,
step=0.01), Output()), _dom_...
<function __main__.resPlot(w)>
```

Out[7]:

Det ses ved brug af slideren, at residualerne nærmer sig 0 for alle markederne ved ca.  $w^* = 0,9$ , hvilket derfor clearer markedet

## Opgave 5.d)

- Find alle  $L_i$ ,  $y_i$ ,  $\pi_i$ ,  $c_{i,j}$  og  $f_i$  i løsningen. *Hint: Brug det  $w^*$  fra opgave 5.c), der clearer alle markeder og brug funktionen `optimering` \*.*
- Argumenter for, om løsningen er Pareto-efficient. *Hint: Du behøver hverken kode eller udledninger til at svare på dette.*

```
In [8]: # SKRIV DIN LØSNING HER:
w_star = np.array([0.9, 0.9, 0.9])
L, y, pi, c, f = optimering(p, w_star)
print("L=", L)
print("y=", y)
print("pi=", pi)
print("c=", c)
print("f=", f)
```

```
L= [0.40644211 0.40644211 0.40644211]
y= [0.54869684 0.54869684 0.54869684]
pi= [0.18289895 0.18289895 0.18289895]
c= [[0.18048316 0.18048316 0.18048316]
     [0.18048316 0.18048316 0.18048316]]
f= [0.60161053 0.60161053 0.60161053]
```

Det ses, at det er en Walras ligevægt, og ifølge første velfærdsteorem, så er enhver Walras-ligevægt pareto optimal og altså derfor efficient.