Rapport du Projet d'Application Final
Setting a carbon tax using Lagrange's duality

Edouard Albert-Roulhac, Julien Ajdenbaum, Waël Rahioui, Aurélien Castre

2 juillet 2021

Avant-propos

L'objectif principal de ce projet est la résolution et la compréhension d'un problème récurrent en macro-économie, la minimisation du prix payé par les agents sous des contraintes d'utilité[1], auquel s'ajoute une autre contrainte sur le coût carbone total de l'activité étudiée. On fera alors apparaître une taxe carbone à appliquer aux prix de l'activité de sorte à respecter la contrainte[2].

La première partie de ce rapport expose le cadre choisi pour la résolution du problème. Les deuxièmes et troisièmes parties donnent deux méthodes de résolution en présentant leurs avantages théoriques et pratiques respectifs. La quatrième et dernière partie porte sur l'étude d'une modélisation alternative dans laquelle on cherche à maximiser l'utilité des agents. On s'attachera, dans chacune des parties, à donner une interprétation économique des calculs effectués.

Nous tenons à remercier Olivier Fercoq pour son aide et son suivi attentif pendant ce projet, malgré une première semaine très mouvementée de son côté!

[1] La notion d'utilité économique correspond, de façon intuitive, au bonheur des agents. Pour plus de détails et d'exemples, se référer à (Foudi n.d.)

[2] Pour une présentation exhaustive du problème, se référer à (Fercoq 2021)

1 Construction du modèle

Pour mener à bien l'étude de ce projet, nous souhaitions modéliser une situation qui soit à la fois proche de notre environnement, afin de mieux pouvoir saisir les conséquences de nos conclusions, et à la fois riche en données fiables et réelles, pour pouvoir calibrer correctement notre modèle. C'est donc tout naturellement que nous nous sommes tournés vers l'étude des événements de l'association Kebab Factory Télécom (KFT), qui organise des ventes de kebabs.

En regroupant les données de l'Agence De l'Environnement et de la Maîtrise de l'Énergie (ADEME n.d.) et des différentes factures de KFT, nous avons abouti à ce tableau récapitulatif regroupant toutes les données nécessaires à la modélisation de l'événement :

	Coût carbone par dose (gCO_2)	Prix par dose TTC (€)	Dose (g)
Pain	132.2	0.3	110
Galette	81.54	0.173	90
Viande	3049.2	1.17	140
Viande vegan	180.6	2.28	140
Salade, Tomates, Oignons	155.4	0.168	105
Sauce blanche	88	0.264	40
Sauce industrielle	60	0.0881	25
Canettes	359.7	0.408	330
Frites	415.8	0.195	330

La deuxième étape fut le choix des agents. Nous avons opté pour 4 agents différents que l'on peut voir comme des catégories de population. La consommation de chaque agent représentera alors la consommation moyenne de cette catégorie, ce qui autorise l'achat de 0.6 doses de pain, par exemple. Ainsi, l'agent 1 ne voudra pas manger de viande vegan, l'agent 2 sera vegan, l'agent 3 sera ni l'un ni l'autre et l'agent 4 aura davantage faim.

Il a enfin fallu définir les fonctions d'utilités des différents agents. Pour coller au mieux à l'intuition, il faut que ces fonctions soient concaves (on augmente moins son bonheur en consommant 20 tablettes de chocolat au lieu d'une qu'en en consommant une au lieu de zéro). De plus, nous les munissons de deux paramètres ajustables de sorte qu'elles différent en fonction des agents. Ainsi, en notant A l'ensemble des agents, I celui des produits et $X = (x_{a,i})_{(a,i)\in A\times I}$ la matrice de la quantité de produit acheté par chaque agent on définit la fonction d'utilité de l'agent a comme :

$$U_a(X) = \sum_{i\in I} \left(r_{a,i}\sqrt{x_{a,i} + b_{a,i}} \right) - U_a(0)$$

Le paramètre $R = (r_{a,i})_{(a,i)\in A\times I}$ permet d'ajuster la quantité d'utilité qu'un produit donné apporte à un agent donné, et le paramètre $B = (b_{a,i})_{(a,i)\in A\times I}$ permet de modifier la pente à l'origine de l'utilité. Ces différents paramètres ont été choisis de façon à obtenir le résultat le plus proche de la réalité observée lorsque la contrainte carbone est absente et que les utilités doivent être supérieures à 1:

	r1	b1	r2	b2	r3	b3	r4	b4
Pain	0.1	0.01	0.0692	0.01	0.11	0.01	0.09	0.01
Galette	0.1	0.00005	0.0692	0.00005	0.11	0.00005	0.09	0.00005
Viande	0.8	0.007	0.508	1.5	0.8	0.1	0.64	0.1
Viande vegan	0.7	1	0.9	0.001	0.8	0.1	0.64	0.1
Salade, Tomates, Oignons	0.11	0.1	0.075	0.1	0.12	0.1	0.096	0.1
Sauce blanche	0.15	0.2	0.1	0.2	0.16	0.2	0.096	0.2
Sauce industrielle	0.07	1	0.04	0.5	0.08	1	0.064	1
Canettes	0.24	0.2	0.158	0.2	0.25	0.2	0.2	0.2
Frites	0.14	0.1	0.11	0.5	0.15	0.1	0.12	0.1

Pour rapprocher les consommations des agents de la véritable consommation dans les événements KFT sans taxe carbone, nous sommes partis de valeurs assez cohérentes pour r et b. Cependant, nous n'arrivions pas à obtenir précisément la consommation souhaitée. Nous avons donc écrit l'algorithme suivant qui modifie aléatoirement les valeurs de r et de b pour se rapprocher progressivement d'une consommation logique. Par exemple, pour l'agent 1, nous avions défini sa consommation logique comme étant : [0.3,0.7,1,0,1,0.5,1,0.5,1]. Avec les valeurs de r et b obtenues par tâtonnement, nous obtenions la consommation suivante : [0.21 0.67 0.86 0. 0.76 0.45 0.27 0.5 0.94] L'algorithme modifie aléatoirement entre 1 et 3 valeurs de r et de b et mesure ensuite la différence entre la consommation obtenue et la consommation optimale (en calculant la norme 2 de cette différence). Si cette mesure est inférieure à la mesure avec les r et b précédents, alors on garde ces nouvelles valeurs. Par changements aléatoires successifs et en faisant des changements de moins en moins importants, on se rapproche peu à peu de la consommation idéale.

A partir des utilités trouvées à la main on obtient une différence de 0.63. Après avoir fait tourner l'algorithme quelques fois, on peut réduire cette différence à $1 * 10^{-5}$ et obtenir une consommation de [0.3 0.7 1. 0. 1. 0.5 1. 0.5 1.].

```
but = [[0.3,0.7,1,0,1,0.5,1,0.5,1]]

def mesureDifference(test,but):
    total = 0
    for i in range(len(test[0])):
        total += abs(test[0][i]-but[0][i])**2
    return total

autre = [[0,0,0,0,0,0,0,0,0]]
nbit = 10
bests = []
dico = {}

```

```
def getDiff(r,b):
    objective = cp.Minimize(cp.sum(X @ p))
    constraints = [centered_Ua(X, 0) >= uamin]
    prob = cp.Problem(objective, constraints)
    res = prob.solve()
    diff = mesureDifference(X.value, but)
    return diff, X.value

pastdiff, pastX = getDiff(r,b)
print("getX" ,np.round(pastX,2))
print('diff', pastdiff)
for i in range(10000):
    rAvant = r.copy()
    bAvant = b.copy()
    n = random.randint(1,10)
    ralentissement = 50
    if(i%100==0):
        print(3 + ((i // ralentissement) + 1))
    for num in range(n):
        j = random.randint(0, 1)
        k = random.randint(0, 8)
        multi = (1 +  (random.random() - 0.5)/ (50))
        # print(multi)
        if(j==0):
            r[k][0] = r[k][0]*multi
        else:
            b[k][0] = b[k][0] * multi
    diff, Xa = getDiff(r,b)
    print(diff)
    if(diff < pastdiff):

        print(r)
        print(b)
        print(multi)
        print(i)
        print("x = ", np.round(Xa, 2))
        print("diff = ", diff)
        print("\n\n\n")
        pastdiff = diff
    else:
        # print("no change")
        r = rAvant.copy()
        b = bAvant.copy()
```

2 Résolution du problème avec cvxpy

Le code python suivant consitue le cœur de l'utilisation du module cvxpy pour la résolution du problème:

```
# --------------- Definition des fonctions d'utilite --------------- #
def Ua(X, a):
    S=0
```

```
    for i in range(nb_of_products):
        S+=r[i][a]*cp.power(X[a][i] + b[i][a], 0.5)
    return S

def centered_Ua(X, a):
    return Ua(X, a) - Ua(np.zeros(X.shape), a)

# --------------- Definition du probleme --------------- #

objective = cp.Minimize(cp.sum(X@p))
constraints = [centered_Ua(X, 0)>=uamin, centered_Ua(X, 1)>=uamin, centered_Ua(X, 2)>=uamin,
    centered_Ua(X, 3)>=uamin,\
                cp.sum(X@c) - C <=0,\
                X[:,0] + X[:,1] >= 1, X[0:,2] + X[0:,3] >= 0.5, X[0:,4]>=0.5, X[0:,5] + X[0:,6]
     >= 0.5, X[0:,7] >= 0.7, X[0:,8] >=1]
prob = cp.Problem(objective, constraints)
res = prob.solve()
```

Les lignes 14 et 15 définissent les contraintes du problème sur l'utilité et la consommation de carbone. La ligne 16 correspond à l'ajout d'une contrainte alimentaire pour éviter que les agents ne prennent plus que des produits à très bas coût carbone au détriment de la cohérence du kebab une fois la contrainte carbone effective. D'autre part, `p` et `c` désignent respectivement les vecteurs de prix et de coût carbone, par dose.

Lorsque l'on résoud le problème sans contrainte carbone, on obtient bien un résultat cohérent (cf. Figure 1). En effet, l'agent 1 ne consomme pas de viande vegan, l'agent 2 ne consomme pas de viande, l'agent 4 mange plus que la moyenne et tous les agents achètent suffisamment pour constituer un kebab.

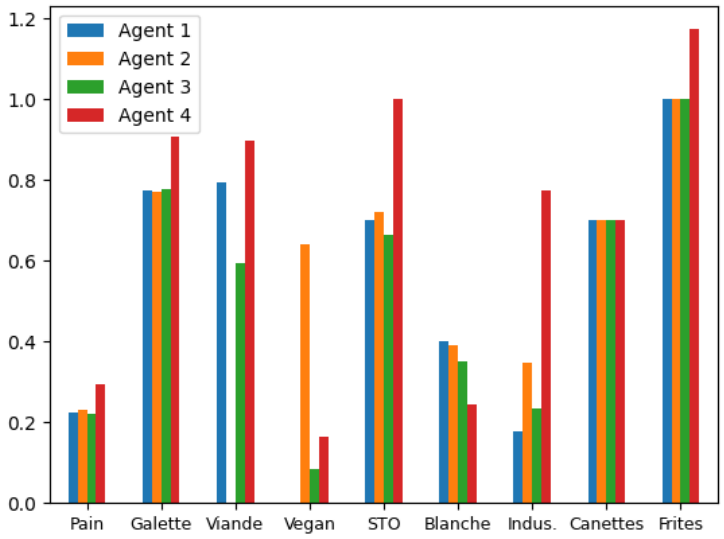

Figure 1: Distribution des achats des agents sans contrainte carbone

L'étude du problème dual (Knowles 2010) nous apprend que chaque contrainte possède une valeur duale. Nous noterons donc λ la valeur duale qui nous intéresse ; celle de la contrainte carbone. Dans notre cas, λ

a une valeur bien concrète: c'est la proportion (en euros par gCO_2) du coût carbone de chaque produit qu'il faudrait ajouter au prix du produit pour que les agents consomment de façon responsable (*i.e.* en respectant la contrainte carbone).

En effet, par définition de λ,

$$\min_{X\geq 0} \sum_{a\in A}\sum_{i\in I} x_{a,i}(p_i + \lambda c_i) - \lambda C = \max_{\lambda'\geq 0}\min_{X\geq 0} \sum_{a\in A}\sum_{i\in I} x_{a,i}p_i + \lambda'\left(\sum_{a\in A}\sum_{i\in I} x_{a,i}c_i - C\right)$$

Or, comme on cherche à minimiser une fonction affine en X, on sait que résoudre le problème dual équivaut à résoudre le problème. Or l'équation de droite correspond exactement au problème dual, ainsi comme λC est une constante on en déduit que la solution X doit être la même pour le problème avec les prix augmentés sans contrainte et le problème avec la contrainte. Autrement dit, la solution du problème avec les prix augmentés respecte la contrainte carbone.

Étudions maintenant l'effet de cette contrainte sur la consommation. En commençant par une contrainte relativement faible (pas plus de 5500 gCO_2 émis), on obtient une taxe qui s'élève à environ 0.18% du coût carbone de chacun des produits (cf. Figure 2). On remarque que la consommationo de viande a drastiquement baissé, ce qui a du sens du fait du grand coût carbone de celle-ci. L'agent 1 compense donc sa frustration dans la sauce pour effacer le goût de la viande vegan qu'il abhorre.

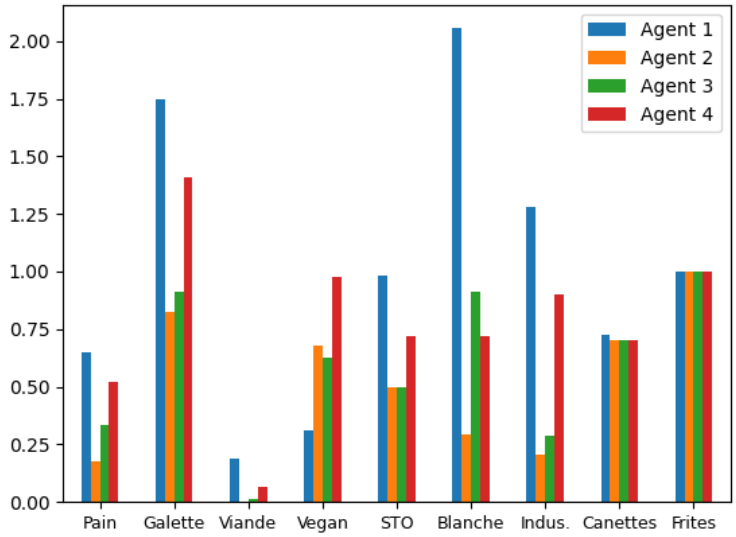

Figure 2: Distribution des achats des agents pour $C = 5500\ gCO_2$, $\lambda \approx 0.0018$

Pour une contrainte plus forte de 4450 gCO_2 on observe les mêmes effets renforcés (cf. Figure 3). Cette fois, la taxe s'élève à plus de 2% du coût carbone.

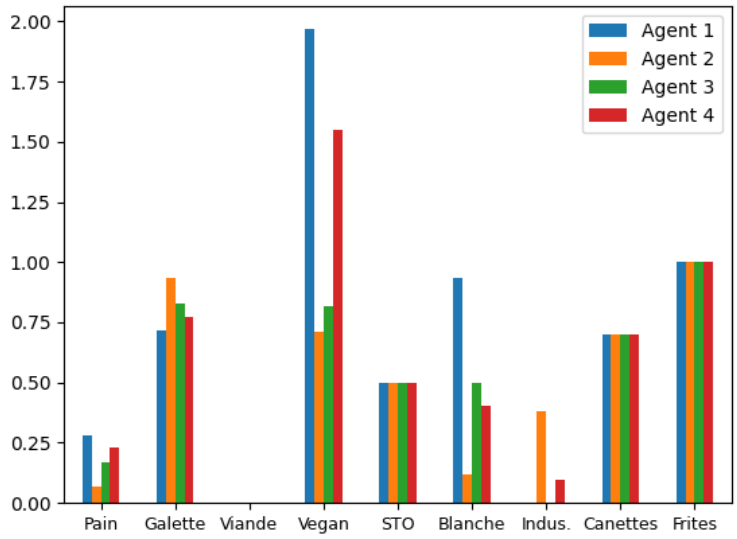

Figure 3: Distribution des achats des agents pour $C = 4450\ gCO_2$, $\lambda \approx 0.021$

Si cette méthode permet de bien appréhender le fonctionnement de la taxe, elle ne permet cependant pas d'imaginer une réelle mise en pratique. En effet, cette méthode impose de connaître parfaitement les fonctions d'utilité des différents agents, ce qui est impossible en pratique. Pour y remédier, nous étudions une deuxième méthode permettant de modéliser l'action d'un gouvernement sur la taxe.

3 Résolution du problème par maximisation de la fonction duale avec la méthode des multiplicateurs de Lagrange

Le principe de la méthode est le suivant: initialement, on observe comment la population consomme, sans contrainte. En utilisant ces données, on définit une valeur de taxe carbone et on regarde comment la population répond. De proche en proche, le gouvernement observe l'effet de l'imposition passée et acutalise la valeur de la taxe de sorte à converger vers une valeur optimale.

Mathématiquement, cela correspond à maximiser la fonction duale (cela donne, dans notre cas, la valeur optimale), qui se trouve être simplement une fonction concave à une dimension (c'est le minimum d'un ensemble de droites affines). Pour résoudre ce problème d'optimisation, nous avons utilisé la méthode des multiplicateurs de Lagrange (Diamond n.d.) implémentée par le code suivant:

```
#On resoud le probleme une premiere fois sans taxe carbone
objective = cp.Minimize(cp.sum(X@p))
constraints = [centered_Ua(X, 0)>=uamin, centered_Ua(X, 1)>=uamin, centered_Ua(X, 2)>=uamin,
    centered_Ua(X, 3)>=uamin,\
               X[:,0] + X[:,1] >= 1, X[0:,2] + X[0:,3] >= 0.5, X[0:,4]>=0.5, X[0:,5] + X[0:,6]
     >= 0.5, X[0:,7] >= 0.7, X[0:,8] >=1]
prob = cp.Problem(objective, constraints)
prob.solve()
```

```
previous = cp.sum(X@p).value

#on utilise le gradient de la fonction duale pour determiner une meilleure valeur de taxe
lprevious = 0
print(cp.sum(X@c).value, C.value)
l = max(0, rho*(cp.sum(X@c) - C).value)
#for some reason, cvxpy despises high values of l
while(l>1000):
    l=l/2

for t in range(MAX_ITERS):
    print("\nITERATION numero", t)
    #on calcule la fonction duale pour la valeur de la taxe actuelle
    objective = cp.Minimize(cp.sum(X@p) + l*(cp.sum(X@c) - C))
    constraints = [centered_Ua(X, 0)>=uamin, centered_Ua(X, 1)>=uamin, centered_Ua(X, 2)>=
    uamin, centered_Ua(X, 3)>=uamin,\
                X[:,0] + X[:,1] >= 1, X[0:,2] + X[0:,3] >= 0.5, X[0:,4]>=0.5, X[0:,5] + X[0:,6]
     >= 0.5, X[0:,7] >= 0.7, X[0:,8] >=1]
    prob = cp.Problem(objective, constraints)
    prob.solve()
    next = (cp.sum(X@p) + l*(cp.sum(X@c) - C)).value

    l = max(0, l + rho*(cp.sum(X@c) - C).value)
    while(l>1000):
        l=l/2

    #condition pour la recherche lineaire
    if(previous - next > 10**(-2)*cp.abs(cp.sum(X@c).value - C).value):
        print("BAD IDEAAA! previous - next :", previous - next)
        rho = rho/2
        l=lprevious
    #si le saut a apporte une amelioration, on actualise la valeur de previous
    previous = next
```

La variable `rho` qui, dans le fichier python, vaut initalement 1 caractérise la longueur des sauts que l'on fait à chaque itération. Pour converger vers la valeur optimale, il existe deux méthodes réalisables qui consistent toutes les deux à diminuer progressivement la valeur du saut. La première consiste à diminuer la taille du saut à chaque itération, indépendemment de la valeur obtenue. Cette méthode possède l'inconvénient de converger très lentement, ce n'est donc pas celle-ci que nous choisissons. La deuxième consiste à regarder la valeur de la fonction duale avant et après le saut (ce sont les variables `previous` et `next` du code). Si le saut n'a pas permis d'augmenter la fonction duale, c'est qu'il est inutile donc on revient d'un saut en arrière (`lprevious`) et on divise la taille du saut par 2.

Pour un gouvernement, l'avantage de la première méthode est d'éviter de revenir sur ses décisions et ainsi d'éviter la perte de crédibilité. Cependant, la deuxième méthode assure une plus grande fiabilité sur le résultat final. Par exemple, dans le cas de la taxe légère de 5500 gCO_2, on obtient un résultat précis à 5 chiffres significatifs près au bout de 50 itérations (50 années si on décide de renouveler la taxe chaque année): 0.17934% du coût carbone.

4 Une modélisation alternative

Une autre modélisation des comportements des agents consiste à supposer qu'ils chercher à maximiser leur utilité (leur bonheur) sous une contrainte de budget. Le code suivant en donne une implémentation:

```
1 objective = cp.Maximize(0.2*centered_Ua(X, 0) + 0.1*centered_Ua(X, 1) + 0.4*centered_Ua(X, 2)
      + 0.3*centered_Ua(X, 3))
2 constraints = [X[0]@p - P0 <=0, X[1]@p - P1 <=0, X[2]@p - P2 <=0, X[3]@p - P3 <=0,\
3                  cp.sum(X@c) - C <=0,\
4                 X[:,0] + X[:,1] >= 1, X[0:,2] + X[0:,3] >= 0.5, X[0:,4]>=0.5, X[0:,5] + X
      [0:,6] >= 0.5, X[0:,7] >= 0.7, X[0:,8] >=1]
5 prob = cp.Problem(objective, constraints)
6 res=prob.solve()
```

Plusieurs choses sont à noter ici. Tout d'abord, il convient de définir ce que l'on entend par maximiser l'utilité. Nous avons choisi de maximiser la moyenne des utilités de chaque agent pondérée par la proportion indicative de la population qu'ils représentent au sein de Télécom (ligne 3). Cela permet, en théorie, d'accorder autant d'importance au bonheur de chacun. De façon réciproque, cela modélise une société dans laquelle chacun est également préocuppé par son propre bonheur.

Ensuite, un changement est à noter à la ligne 2. Nous avons choisi d'accorder une contrainte de budget spécifique à chaque agent, l'agent 1 et l'agent 4 ayant des désirs plus coûteux à assouvir. Cela montre aussi une dérive de ce modèle: si un agent est très riche, sa contrainte sera très facile à respecter et ce sera son utilité qui sera favorisée dans la somme car ce sera la plus simple à obtenir. Cela peut être contré par les pondérations, jusqu'à un certain point.

Enfin, il faut remarquer que cette méthode ne fournit pas de taxe carbone exploitable. En effet, λ représente ici la quantité d'utilité qu'il faudrait fournir aux agents pour qu'ils consomment de façon responsable, et elle s'exprime en utilité par gramme de CO_2. Cette donnée est très délicate à manipuler car aucun lien évident ne peut être construit entre l'utilité et les euros. De plus, cela suppose que l'on pourrait injecter de l'utilité dans les agents sans produire un gramme de CO_2, ce qui relève de l'utopie.

Références

ADEME (n.d.). *Base carbone de l'ADEME*. URL: https://www.bilans-ges.ademe.fr/en/basecarbone/donnees-consulter.

Diamond (n.d.). *Method of multipliers*. URL: https://www.cvxpy.org/examples/applications/MM.html.

Fercoq, Sowunmi (2021). *Setting a carbon tax using Lagrange's duality*.

Foudi (n.d.). *Cours de micro-économie - introduction*.

Knowles (2010). *Lagrangian Duality for Dummies*.