Optimisation des restaurants de Jussieu

(ARE dynamic)

Nicolas Alam – Mehdi Lahboubi – Louis Bresson – Félix Lopez

Problématique :

Comment améliorer le CROUS de Jussieu ?

Hypothèse principale :

Le restaurant le moins cher est le plus attractif.

Objectif :

Evaluer la satisfaction globale du groupe.

Paramètres (restaurants) :

- Nombre de restaurants : fixé à 3 (CROUS, five pizza, crêpes)

Pour chaque restaurant :

- Rapidité su service
- Prix
- Distance aux autres restaurants

Paramètres (client) :

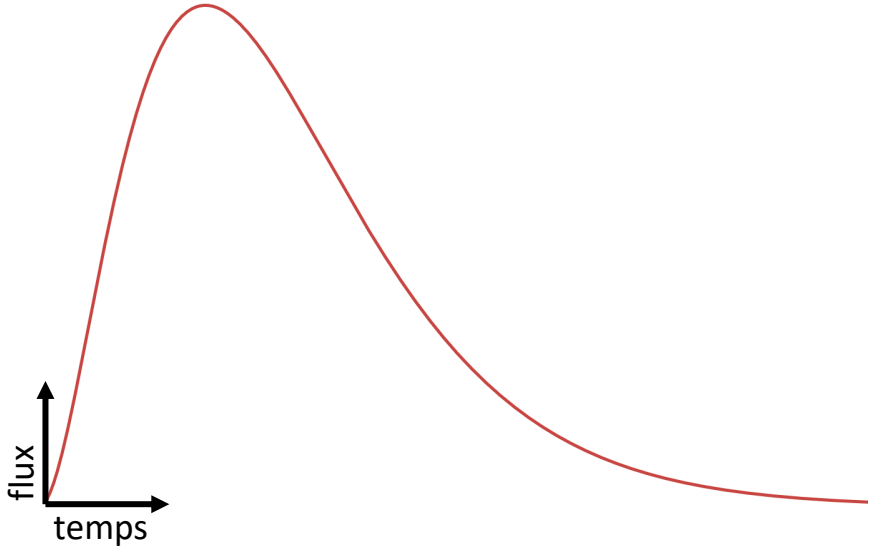
$$f_{a,b}(x) = ax^2 e^{-\frac{x}{b}}$$

$$c = \max_{\mathbb{R}+} f_{a,b} = 4ab^2e^{-2} \qquad d = \int_{\mathbb{R}+} f_{a,b} = 2ab^3$$

- Nombre total et flux de clients :

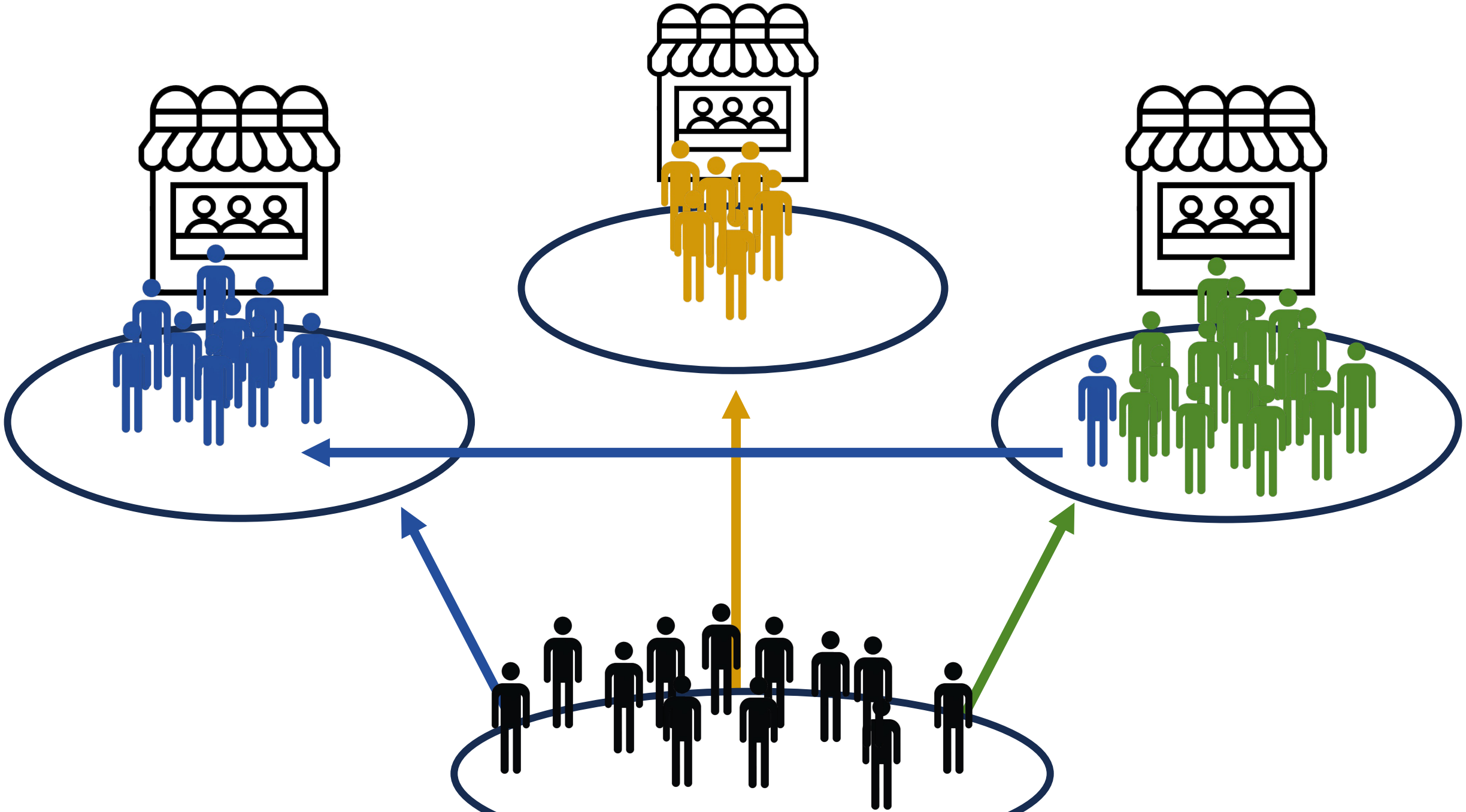
Pour chaque client :

- Patience
- Budget
- Préférences



$$a = \frac{c^3e^6}{16d^2} \qquad b = \frac{2d}{ce^2}$$

$$f_{(c,d)}(x) = \frac{c^3e^6}{16d^2} xe^{-\frac{ce^2x}{2d}}$$

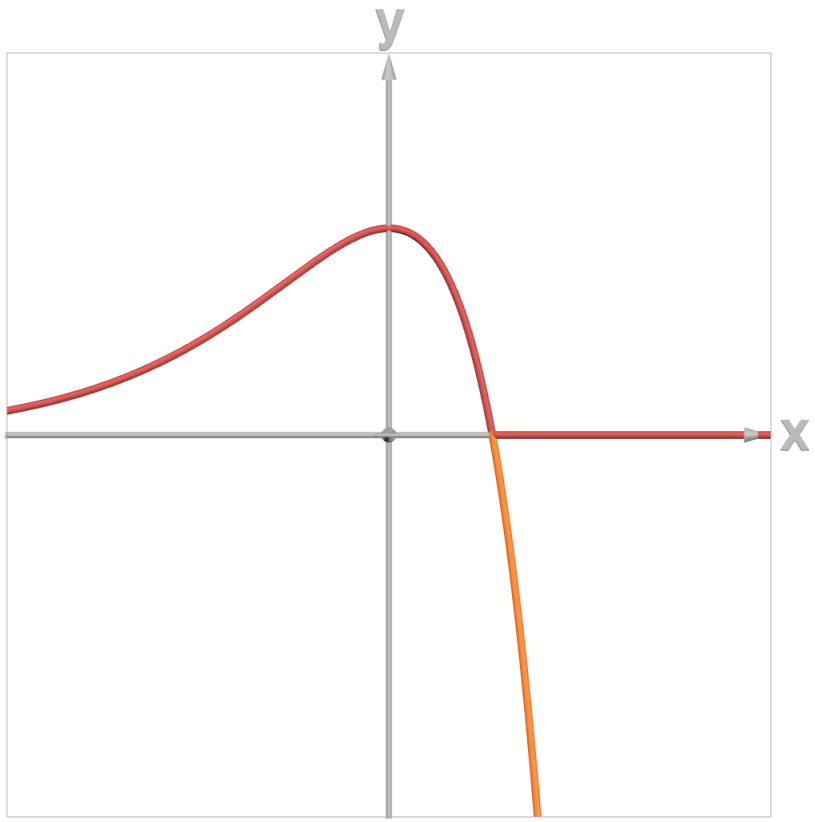


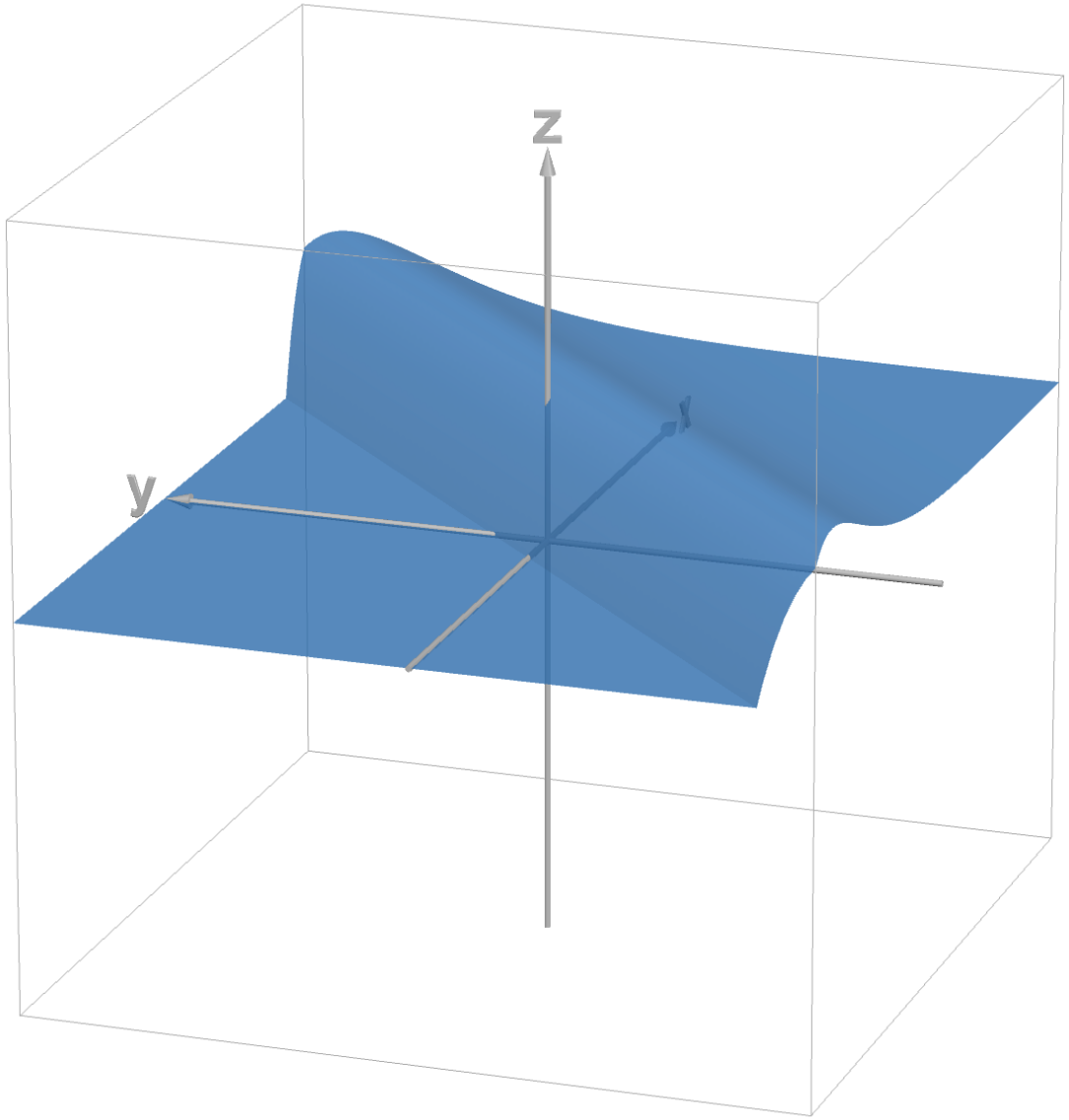
Classe Client : attributs et méthodes

```python
class Client:
    def __init__(self, id, restaurant, patience, budget, pref_lists, waiting=True):
        self.id = id
        self.restaurant = restaurant
        self.patience = patience
        self.budget = budget
        self.pref_lists = pref_lists
        self.waiting = waiting
        self.current_max_appeal = 0

    def appeal_byprice(self, restaurant):
        p = restaurant.avg_price
        b = self.budget
        axs = p-b+.3
        val = (1-2*axs)*np.exp(2*axs)
        return max(0, (val+1)/2 )

    def appeal_byqueue(self, restaurant):
        r = restaurant.rank(self.id)+1
        e = restaurant.eff
        p = self.patience
        return np.exp(-r/e/p)

    def appeal_bydistance(self, restaurant):
        if self.restaurant:
            return np.exp( -self.restaurant.walking_time[restaurant.id]/2 )
        else:
            return np.exp( -restaurant.walking_time[-1])

    def appeal_byprefs(self, restaurant):
        return self.pref_lists[restaurant.id] * 5

    def appeal(self, restaurant):
        bp = self.appeal_byprice(restaurant)
        bq = self.appeal_byqueue(restaurant)
        bd = self.appeal_bydistance(restaurant)
        bpr= self.appeal_byprefs(restaurant)
        app = 6*bp + 4*bq + 1*bd + 1*bpr
        return app

    def bestRestaurant(self, restaurants):
        appeals = [self.appeal(restaurant) for restaurant in restaurants]
        self.current_max_appeal = max(appeals)
        return restaurants[ max(range(len(appeals)), key=appeals.__getitem__) ]
```

Fonction d’attrait par le prix

Fonction de référence : $f(x) = \max(0, (1-2x)e^{2x})$



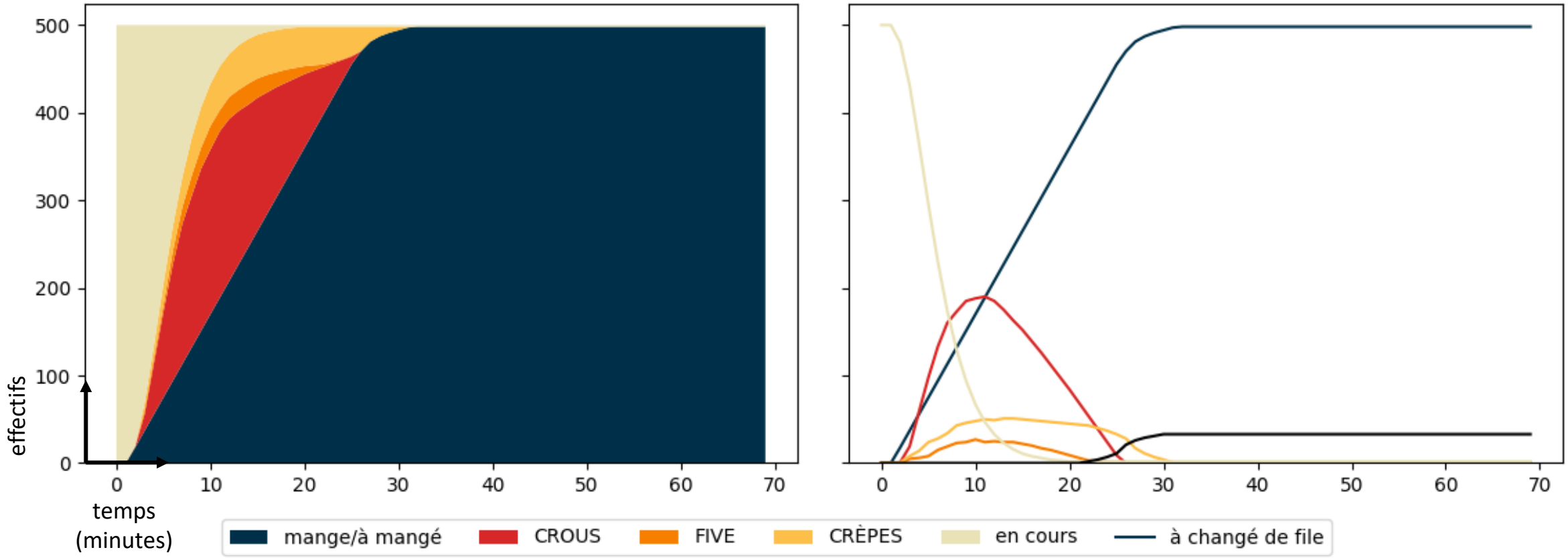


X : budget du client

Y : prix moyen dans le restaurant

Z : attractivité due au prix

Hypothèse principale : Le restaurant le moins cher est le plus attractif.



CROUS : 77.6% FIVE : 15.3% CRÊPES : 7.1% Global Satisf. : 5.223

Problématique : Comment améliorer le CROUS de Jussieu ?

$f($ efficacité $) =$ prix repas
(nombre d'employés)

$$f(x) = 7 - \frac{7000 - 60x}{1600}$$

7	=	coût d'un repas
1600	≈	nombre de repas
7000	≈	budget quotidien
60	=	SMIC/jour (brut)