Comparaison méthodes classiques et adaptatives (UCB)

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
# fonctions générales
# [ .... ] (cf listings précédents)
```

In [4]:

```
actions_q = [0.82, 0.92, 0.87, 0.86, 0.84, 0.83]
#actions similaires à celles de l'essai clinique de l'AVC
#mais plus dispsersées afin d 'avantager UCB
```

On code deux fonctions qui implémentent respectivement des essais cliniques sur un problème à 6 traitements avec les probabilités `actions_q` définies plus haut. Ces deux fonctions renvoient le nombre de patients vivants à la fin de l'essai clinique.

Pour simplifier, on suppose qu'un patient, lorsqu'il prend un traitement, guérit de la maladie ou meurt.

In [5]:

```
def essai_randomise():
    """
    Implémente un essai clinique randomisé (distribution aléatoire uniforme
    des traitements)
    sur 20000 patients, avec les probabilités définies plus haut.

    Renvoit le nombre de patients vivants parmi les 20000 à la fin
    de l'essai clinique.
    """

    vivants = 20000

    for t in range(1, 20000):
        a = np.random.randint(len(actions_q)) #action aléatoire, car essai rando
misé
        r = generer_reponse(actions_q, a)

        if r == 0:
            vivants = vivants-1

    return vivants
```

In [6]:

```
def essai_ucb(n_patients, p_mort, jours_attente):
    """
    Implémente un essai clinique adaptatif par la méthode d'UCB
    (version groupé).

    n_patients est le taille des groupes de patients
    (n_patients = 1 revient à UCB classique).

    jours_attente est le nombre de jours qu'il est nécessaire d'attendre
    entre l'administration d'un traitement à un patient
    et les effets du traitement sur celui-ci
    (supposé le même pour les 6 traitements)

    p_mort est la probabilité quotidienne qu'a un patient
    de mourir de la maladie.

    Renvoit le nombre de patients vivants parmi les 20000 à la fin de l'essai cl
inique.
    """

    non_traites = 20000
    vivants = non_traites

    actions_Q = [0, 0, 0, 0, 0, 0]
    actions_U = [0, 0, 0, 0, 0, 0]
    actions_N = [1, 1, 1, 1, 1, 1]

    actions_compteurs = [[0], [0], [0], [0], [0], [0]]

    t = 1
    while non_traites > 0:
        #le dernier groupe peut être plus petit
        #en fonction des patients restants à traiter
        if non_traites < n_patients:
            n_patients = non_traites

        #calcul de U_t(a) pour toutes les actions a
        for a in range(len(actions_q)):
            actions_U[a] = np.sqrt(2 * np.log(t) / actions_N[a])

        #on choisit l'action à administer au groupe
        a = indice_max(ajouter_listes(actions_Q, actions_U))

        for i in range(n_patients):
            actions_N[a] += 1

            r = generer_reponse(actions_q, a)

            #moyenne incrémentale
            actions_Q[a] = actions_Q[a] + (1/actions_N[a]) * (r - actions_Q[a])

            actions_compteurs[a].append(actions_compteurs[a][-1] + 1)
            for b in range(len(actions_q)):
                if b != a:
                    actions_compteurs[b].append(actions_compteurs[b][-1])

            #on a traité un patient
            non_traites = non_traites - 1
```

```
            #on enlève le patient des vivants si il meurt
            if r == 0:
                vivants = vivants - 1

        #on doit ici simuler le temps d'attente de la réponse du traitement
        #sur les patients du groupe.

        #chaque jour qui s'écoule, chacun des patients non traités
        #peut mourir avec une probabilité p_mort
        for d in range(jours_attente):
            morts = np.random.binomial(non_traites, p_mort)
            non_traites = non_traites - morts
            vivants = vivants - morts

        t = t + n_patients

    return vivants
```

In [7]:

```
def adaptatif_meilleur(n_patients, p_mort, jours_attente):
    """
    Appelle les deux fonctions définies plus haut,
    et retourne 1 si l'essai clinique adaptatif est meilleur
    que l'essai clinique randomisé, 0 sinon.

    On effectue pour cela 5 essais pour avoir un réponse plus fiable.
    """

    adpatatif_meilleur_compteur = 0

    for i in range(5):
        if essai_ucb(n_patients, p_mort, jours_attente) > essai_randomise():
            adpatatif_meilleur_compteur += 1

    return round(adpatatif_meilleur_compteur/5)
```

In [8]:

```
# à n_patients fixé, on va afficher en 2 dimensions les zones pour lesquelles
# les essais UCB sont meilleurs que les essais randomisés.

# on définit pour cela les linspaces nécessaires avec numpy
# et l'on crée alors une meshgrid, ce qui va permettre d'appeler
# la fonction adaptatif_meilleur en chaque combinaison de points (jours_attente,
p_mort)

n_patients = 500

jours_attente_min = 1
jours_attente_max = 15

p_mort_min = 10e-9
p_mort_max = 10e-4
p_mort_nombre_points = 20

def f(x, y):
    return adaptatif_meilleur(n_patients, y, x)

jours_attente = np.linspace(jours_attente_min, jours_attente_max,
                            jours_attente_max - jours_attente_min + 1,
                            dtype=np.int64)
p_mort = np.linspace(p_mort_min, p_mort_max, p_mort_nombre_points)
X, Y = np.meshgrid(jours_attente, p_mort)

f_vec = np.vectorize(f)
Z = f_vec(X, Y)
```

In [12]:

```
plt.contourf(X, Y, Z, levels=1, colors=['red', 'blue'], alpha=0.2)

plt.title('n_patients= ' + str(n_patients))
plt.ylabel('p_mort')
plt.xlabel('jours_attente')

plt.savefig('comparaison_' + str(n_patients) + '_patients', dpi=300)
```