Rapport Projet 2A : Le Vote Unique à Transfert Automatique

Auteurs :
Rody Assaf
Valentin Chandor

Tuteur :
Mr. Rémi Peyre

Remerciements

Nous tenons à remercier notre tuteur, Mr. Rémi Peyre, qui a su nous aiguiller tout au long du projet. Son investissement et ses conseils nous ont permis de mieux appréhender le sujet. Son aide fut précieuse dans les moments les plus délicats et nous a permise de pouvoir mener le projet à bout.

Table des matières

1 Introduction

Aujourd'hui, même si le principe de l'élection au suffrage universel (droit de vote accordé à tous les citoyens majeurs) fait l'unanimité dans les démocraties représentatives, il n'en va pas de même pour le choix du mode de scrutin. En effet, le mode de scrutin permettant le passage du décompte des voix à la désignation des élus est souvent sujet à débat. Il apparaît comme un savant mélange où interviennent notamment l'histoire politique nationale, les besoins de représentativité ou encore l'opinion des partis. Par exemple, la France a connu plusieurs modes de scrutin. Entre 1791 et 1871, c'est le scrutin de liste qui prévalait. A partir de 1871, la majorité monarchiste à l'Assemblée nationale institue le système majoritaire à deux tours. Plus tard, en 1945, le Général de Gaulle fait adopter le scrutin à la proportionnelle ; en 1958, le régime de la Vème République revient au scrutin uninominal majoritaire pour l'élection des députés. Chacun de ces différents modes de scrutins possède ses avantages et inconvénients. Nous nous focaliserons dans notre étude sur l'élection d'une assemblée. [Winb]

Tout d'abord, dans le scrutin uninominal à deux tours, la réussite au premier tour est conditionnée par l'obtention d'une majorité absolue des voix, avec parfois l'obligation de réunir un nombre minimal d'électeurs inscrits. Faute d'avoir atteint ce seuil, un second tour est organisé. Son accès est réglementé et il met aux prises les deux candidats les mieux placés au premier tour (scrutin présidentiel français) et les candidats ayant recueilli un nombre minimum de voix ou un certain pourcentage des inscrits (scrutin législatif français). La possibilité de conclure des alliances pour le second tour lisse les distorsions : les petits partis peuvent s'entendre avec d'autres pour obtenir des élus en échange d'un report de voix. En revanche, ceux qui ne souscrivent pas d'alliance sont souvent privés de toute représentation. Ce mode de scrutin, présente l'avantage de désigner une majorité stable, en mesure de gouverner. Cependant, son principal défaut est de ne pas conférer à l'assemblée désignée une représentation fidèle du corps électoral.

Le scrutin de liste, quant à lui, est un système de comptage de voix reposant sur le principe suivant : chaque parti propose une liste de candidats, l'électeur choisit une liste, et la liste ayant obtenu une majorité de voix est la liste gagnante, obtenant tous les sièges. On peut avoir un scrutin majoritaire simple, où la liste est élue à la majorité relative au premier tour, ou bien un scrutin majoritaire à deux tours, où la liste doit être élue à la majorité absolue au premier tour ou relative au second tour. Les désavantages majeurs de ce type est scrutin est qu'il avantage les grands partis en plus de ne pas laisser à l'électeurs le choix de la liste qui représentera son parti.

Pour remédier à ces inconvénients, il existe un type de scrutin appelé scrutin à vote unique transférable(VUT). Le système de scrutin à vote unique transférable a été conçu pour répondre aux deux objectifs suivants : -chaque électeur doit pouvoir choisir son ou ses candidats, sans s'en remettre au choix d'un parti

-le nombre d'élus doit correspondre à une répartition proportionnelle.

Le VUT vise à assurer la représentation proportionnelle tout en écartant l'influence des partis qui, de par leur fonctionnement propre, peuvent composer des listes ne coïncidant pas avec les souhaits des citoyens, par exemple en plaçant en tout début de liste des candidats bien en cour mais peu appréciés des électeurs. Il respecte ainsi pleinement la volonté populaire contre les logiques d'appareils. Le vote unique transférable est actuellement utilisé en Irlande et en Nouvelle-Zélande

entre autres, pour des élections locales. Il procure le pouvoir au citoyen de pouvoir composer sa propre liste de choix. Cela signifie que la personne qui vote, compose une liste de candidats qui sont ordonnés selon les préférences de l'électeur. Lors de l'élection ou élimination d'un candidat, les voix d'un électeur peuvent se reporter vers ses vœux suivants. La première conséquence de ce type de vote est qu'il n'y a qu'un tour néanmoins un inconvénient majeur est qu'il y a potentiellement autant de listes possibles que de votants. C'est donc dans cette logique qu'intervient le vote unique à transfert automatique (ou VUTA), qui est une amélioration du vote unique transférable étant donné que ce sont les candidats qui constituent leur propre liste.

Afin de réaliser ce projet, notre tuteur, Mr. Peyre nous a fixé 2 grands axes. Le premier étant l'implémentation informatique de ce système puis, la modélisation des choix des électeurs par un modèle mathématique élégant. Dans ce rapport, nous allons tout d'abord expliquer le principe de ce vote puis, détailler les différentes étapes de notre implémentation pour enfin introduire notre modélisation mathématique.

2 Le Vote Unique à Transfert Automatique

Le vote unique à transfert automatique se décompose en deux phases : une première phase que l'on appellera **Phase d'Allocation** puis, une seconde phase dite d'**Élimination**. Dans cette partie nous expliquerons les deux phases qui seront éclairées par des exemples.

2.1 Le principe du VUTA

Soient N candidats à une election qui aboutira à $nbre\_sieges$ candidats élus. De plus, on suppose qu'il y a $nbre\_electeurs$ électeurs qui participent à cette élection. Ces trois chiffres définissent un seuil, le quorum, qu'un candidat devra atteindre s'il veut être élu. On a alors :

$$quorum = \frac{\text{nbre\_electeurs}}{\text{nbre\_sieges}} \tag{1}$$

De plus, chaque électeur votera pour une liste de candidats de transfert (par souci de clarté nous appellerons ce type de liste une liste de transfert), liste dans laquelle le candidat ayant composé cette liste se place en premier, suivi d'autres candidats classés suivant ses affinités ou préférences. Lors du dépouillement, on décomptera le nombre de voix associé à chaque liste de candidat de transfert. Pour déterminer si un candidat est élu, il faut regarder à tout instant si son nombre de voix qui lui est alloué dépasse le quorum. Pour cela, on procède par les étapes d'**Allocation** et d'**Élimination**.

2.2 Phase d'Allocation

2.2.1 Position du Problème

Supposons qu'à un instant t (le temps représente la progression de l'allocation), le nombre de voix allouées à un candidat est égal au quorum. Il n'est donc plus nécessaire d'allouer plus de voix à ce candidat sachant qu'il est déjà élu. C'est ici qu'intervient la notion de liste de transfert.

2.2.2 Principe de la phase d'allocation

Nous sommes toujours dans la situation où un candidat atteint le quorum. Les voix qui lui ont été attribuées vont alors ensuite être destinées aux candidats qui se trouvent après lui dans sa liste de transfert. On peut ainsi faire une analogie simpliste avec des vases que l'on remplirait. La contenance d'un vase est représentative du quorum. Dès lors qu'un vase est plein, le candidat est élu. On peut alors représenter une liste de transfert comme une suite de vases communicants.

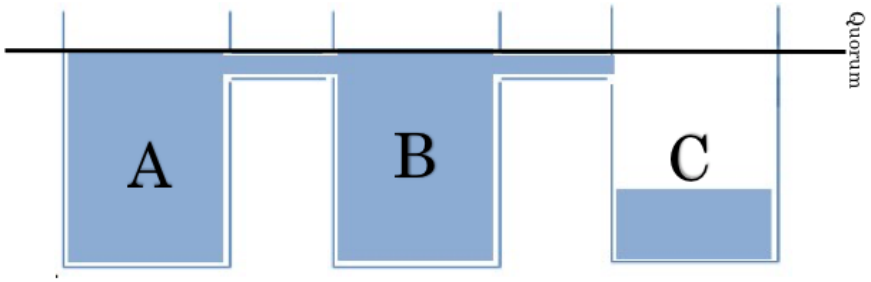

FIGURE 1 – Représentation sous la forme de vases communicants de la liste de transfert du candidat A : [A,B,C]

On matérialise ainsi ci-dessus le principe du transfert de voix lors de la phase d'allocation.

2.3 Phase d'élimination

2.3.1 Position du problème

Supposons qu'à l'issue de la phase d'Allocation, on a réussi à élire $nbre\_e < nbre\_sieges$ candidats. Il faut donc élire les $nbre\_sieges$-$nbre\_e$ restants. Toutefois, on ne peut plus utiliser le mécanisme de l'**Allocation** car, toutes les voix ont été provisoirement attribuées. Il faut donc procéder à la phase d'**élimination**.

2.3.2 Principe de la phase d'élimination

Supposons qu'il reste $nbre\_sieges$-$nbre\_e$ candidats à élire après la phase d'Allocation. On va donc procéder de la manière suivante. Le candidat ayant le moins de voix est éliminé. Ses voix se reportent alors vers le candidat qu'il a placé le plus haut dans sa liste de transfert. On procède donc ainsi jusqu'à ce que les $nbre\_sieges$ soient pourvus.

Dans la suite de ce rapport nous allons expliquer comment nous avons implémenté le VUTA sous Python.

3 Le VUTA en Python, une première implémentation

Nous allons, dans cette partie expliquer comment nous avons décidé dans un premier temps d'aborder le VUTA sous Python. En effet, une version plus complète tenant compte de toutes les subtilités du VUTA sera présentée par la suite. Par souci de clarté, les variables seront en italiques, les noms de fonctions en gras et les arguments de fonction entre guillemets.

3.1 Quelques préliminaires

Nous avons, dans la première partie de notre projet, cherché à coder le processus de vote unique à transfert automatique en essayant de représenter par des variables et objets les différents acteurs et composantes permettant de mener le vote à terme.

Tout d'abord il est clair que le choix judicieux pour représenter : le nombre de votants, le nombre de candidats à l'élection, le nombre de sièges à pourvoir et le nombre de voix nécessaires pour être élu, était des variables entières que nous avons nommées respectivement *nbre_electeurs*, *nbre_candidat* , *nbre_sieges* et quorum.

Ensuite il faut bien entendu représenter les listes de transfert de chaque candidat et les voix des votants. Nous avons décidé de représenter ces deux composantes par des listes, respectivement, *choix_ electeurs* et *LT*. L'intérêt d'utiliser une liste de listes pour *LT* est qu'il est facile de retirer les candidats vainqueurs ou bien d'éliminer de toutes les listes de transfert en le faisant par compréhension.

Finalement, les dictionnaires sont une structure qu'il nous a semblé pertinent d'introduire. Au lieu d'héberger des informations dans un ordre précis comme les listes, un dictionnaire associe chaque objet contenu à une clé. Un dictionnaire est donc un objet conteneur d'autres objets dans lequel il n'y a aucune structure ordonnée. Pour accéder aux objets contenus, on utilise les clés qui leur sont associées. Nous reviendrons plus en détail, dans les parties qui suivent, sur le rôle des dictionnaires dans notre programme.

3.2 Fontions Auxiliaires

3.2.1 Les dictionnaires

Nous avons, dans notre code utilisé régulièrement des dictionnaires car, il est naturel, lors d'un vote d'associer des scores à des candidats et des noms à des listes de candidats.

```
1 def def_dic_tete_de_liste(LT,noms_des_listes_de_candidats):
2     d = dict()
3     for k in range(len(noms_des_listes_de_candidats)):
4         d[noms_des_listes_de_candidats[k]] = LT[k][0]
5     return d
```

La fonction **def_dic_tete_de_liste** crée un dictionnaire qui, à une liste de transfert, associe le nom du candidat en tête de cette liste. Cela fait écho à notre supposition raisonnable qu'un candidat se place en tête de sa liste de transfert. L'argument "*LT*" est la liste des listes_de_candidats_de_transferts, et l'argument "*noms_des_listes_de_candidats*" est la liste des noms respectifs de ces listes_de_candidats_de_transferts.

```
def def_dic_candidat_indice(noms_des_listes_de_candidats):
    d = dict()
    for k in range(len(noms_des_listes_de_candidats)):
        d[noms_des_listes_de_candidats[k]] = k
    return d
```

def_dic_candidat_indice crée un dictionnaire qui, à un nom de liste_de_transferts, associe un numéro : l'identifiant. L'argument "*noms_des_listes_de_candidats*" est la liste des noms de listes_de_candidats_de_transferts

```
def invertdic(dictionnaire):
    return dict([(v,k) for k,v in dictionnaire.items()])
```

La fonction **invertdict** produit le dictionnaire réciproque d'un dictionnaire "*dictionnaire*" dont toutes les valeurs sont distinctes, les valeurs devenant clés et vice-versa. Ceci nous permettra par la suite, en inversant les dictionnaires cités ci-dessus, de retrouver la liste de transfert grâce à son nom et grâce au numéro de la liste de transfert, de lui associer son nom.

3.2.2 Les fonctions de comptage/dépouillement

```
def comptage_list(choix_electeurs,dic_indice_candidat):
    Lcompt = [0 for k in range(len(LT))]
    for k in choix_electeurs:
        Lcompt[dic_indice_candidat[k]]+=1
    return np.array(Lcompt)
```

La fonction **comptage_list** réalise le dépouillement, i.e. elle renvoie le nombre de voix associé à chaque liste. On en aura besoin pour attribuer les voix et déterminer les vainqueurs.

```
def comptage_pers(LT,resultat_allocation_liste_candidats):
    n = len(LT)
    Lc = [0 for k in range(n_candidat)]
    if len(LT[0])!=0:
        for k in range(n):
            v = LT[k][0]
            liste = dic_tete_de_liste_inv[v]
            rang = dic_candidat_indice[liste]
            Lc[rang] = Lc[rang] + resultat_allocation_liste_candidats[k
    ]
    return Lc
```

La fonction **comptage_pers** prend en entrée la liste "*LT*", liste des listes_de_candidats_de_transferts, et la liste "*resultat_allocation_liste_candidats*". Cette dernière entrée est obtenue grâce à la fonction **comptage_list**. On s'en sert pour actualiser le nombre de voix des candidats après suppressions successives des différents candidats dans les itérations. À $t = 0$, la liste issue de **comptage_pers** est identiquement égale à la liste produite par **comptage_list**.

En revanche, plaçons nous à l'instant $t-1$ où les candidats sont [A,B,C,D,E] et E déjà élu. À cet instant, on suppose LT = [[A,B,D,C] , [B,A,C,D] , [C,B,A,D] , [D,B,A,C]]. Imaginons que D dépasse *quorum*. À l'instant t , LT = [[A,B,C] , [B,A,C] , [C,B,A] , [B,A,C]]. On remarque que B est en tête de deux listes. La

fonction **comptage_pers** additionnera pour le candidat B, les voix restantes de la liste LB et de la liste LD. C'est ainsi que l'on réalise l'action de transfert de voix.

3.3 Fonction VUTA

3.3.1 Phase d'allocation

```
def VUTA(LT,resultat_allocation_liste_candidats,n_sieges,quorum):
    liste_vainqueurs = list()
    nombre_de_voix_actualisées = resultat_allocation_liste_candidats

    # PREMIERE PHASE DU DÉPOUILLEMENT: ALLOCATION
    while True:
        maxi,indmax = np.amax(nombre_de_voix_actualisées),np.argmax(
    nombre_de_voix_actualisées) # Nombre de voix et numéro du candidat
    qui a le plus de voix.
        print("le plus grand nombre de voix obtenu par une liste
    electorale est :  ", maxi)
        if maxi>=quorum:
            winner = dic_indice_candidat[indmax]
            print("le vainqueur est donc :",winner)
            liste_vainqueurs.append(winner)
            toquorum = float(maxi-quorum)
            for k in range(len(LT)):
                if LT[k][0] == dic_tete_de_liste[winner]:
                    resultat_allocation_liste_candidats[k] =
    resultat_allocation_liste_candidats[k]/float(maxi)*float(toquorum)

            LT = [[element for element in row if element !=
    dic_tete_de_liste[dic_indice_candidat[indmax]]] for row in LT]

            print("resultat de l'allocation de voix : " ,
    resultat_allocation_liste_candidats)
            print("LT : ",LT)
            nombre_de_voix_actualisées = comptage_pers(LT,
    resultat_allocation_liste_candidats)
            print("nombre_de_voix_actualisées :",
    nombre_de_voix_actualisées)
            print("Liste de Transfert:",LT)
        else:
            break
```

Les entrées sont "*LT*", la liste des liste_de_transferts, "resultat_allocation_liste_candidats", la liste des nombres de voix reçues par chacune de ces *listes_de_candidats_de_transferts*, et "*nbre_sieges*", le nombre de sièges à pourvoir et quorum, le nombre de voix nécessaires à l'élection. La fonction affiche une liste formée de la chaine "les vainqueurs sont :" et de la liste des vainqueurs.

On initialise une liste de vainqueurs vide : *liste_vainqueurs*. On rempliera cette liste jusqu'à obtenir une liste de longueur *nbre_sieges*.

Notre idée est d'utiliser une boucle while de laquelle on ne sortira uniquement lorsque tous les sièges seront pourvus. En réalité, dans la première boucle while, on continue d'itérer tant qu'on peut faire de l'**Allocation**, en d'autres termes, en suivant le processus de transfert pour le report de voix, on peut faire élire des candidats sans en éliminer. C'est pour cette raison que l'on teste, à chaque tour de boucle si le maximum de voix dépasse le quorum. Si ce n'est pas le cas on sort de cette première boucle.

La première étape de la boucle est de regarder le candidat dont la liste a le maximum de voix attribuées : Si le nombre de voix de ce candidat satisfait le quorum on l'ajoute à vainqueur, le candidat est alors stocké dans la variable winner.

— On calcule le nombre de voix supplémentaires par rapport au *quorum*

— On fait évoluer *resultat_allocation_liste_candidats* en réattribuant les voix du candidat suivant la méthode de transfert

On supprime le candidat *winner* de toutes les listes de transfert on fait ensuite évoluer *nombre_de_voix_actualisees* en regardant le nombre de fois où les autres candidats arrivent en première position. La variable "*nombre_de_voix_actualisees*", qui est actualisée au cours du scrutin, est la liste contenant pour chaque candidat, le nombre de voix qui restent à dépouiller et qui se porteraient vers lui en l'état actuel des listes_de_candidats_de_transferts. On veut, en procédant de la sorte, représenter le report des voix selon la règle du VUTA.

Si, à l'issue de la phase d'**Allocation**, tous les sièges ne sont pas pourvus et qu'aucun candidat n'a assez de voix pour atteindre le quorum, on passe à la phase d'élimination.

3.3.2 Phase d'élimination

```
    # SECONDE PHASE DU DEPOUILLEMENT: ELIMINATION
    print("phase d'élimination")
    print("la liste provisoire est :",liste_vainqueurs)
    while len(liste_vainqueurs)<n_sieges or len(LT)>n_sieges:
        mini,indmini = minDiffZero(nombre_de_voix_actualisées)[0],
minDiffZero(nombre_de_voix_actualisées)[1]

        looser = dic_indice_candidat[indmini]
        namelooser = dic_tete_de_liste[looser]

        print("le candidat elimine est : " , namelooser)

        LT = [[element for element in row if element != namelooser] for
 row in LT]

        dicorigine[namelooser] = nombre_de_voix_actualisées[indmini]


        nombre_de_voix_actualisées[indmini] = 0

        k = 0
        while (k<len(LT[indmini]) and dicorigine[namelooser] != 0) :

            candidat_transfert = LT[indmini][k]
            print("l'element transfere est :",candidat_transfert)
            indtransfert = dic_candidat_indice[dic_tete_de_liste_inv[
candidat_transfert]]

            attribution_provisioire = nombre_de_voix_actualisées[
indtransfert] + dicorigine[namelooser]
            print("attribution_provisioire = ", attribution_provisioire
)
            toquorum = attribution_provisioire - quorum
            print("le nombre de voix en trop est de :",toquorum)
            if toquorum>=0:
```

```
                print("condition1")
                liste_vainqueurs.append(dic_tete_de_liste_inv[
    candidat_transfert])
                LT = [[element for element in row if element !=
    candidat_transfert] for row in LT]
                dicorigine[namelooser] = toquorum
                nombre_de_voix_actualisées[indtransfert] = 0
                print("dicorigine : ",dicorigine)


            else:
                print("condition2")
                nombre_de_voix_actualisées[indtransfert] =
    attribution_provisioire
                dicorigine[namelooser] = 0
                k = k+1

            print("dicorigine :",dicorigine)

        print("nombre_de_voix_actualisées :",nombre_de_voix_actualisées
    )


        if len(LT[0]) == n_sieges - len(liste_vainqueurs):
            break


    for k in LT[0]:
        print("Il reste autant de places que de candidats restants")
        liste_vainqueurs.append(dic_tete_de_liste_inv[k])
    return ('les vainqueurs sont :', liste_vainqueurs)
```

La seconde phase dite d'**Élimination** est contenue dans une boucle while dont on ne sortira que s'il n'y a plus de sièges à pourvoir ou, s'il reste autant de places libres que de candidats restants dans la liste de liste de transfert LT. Premièrement nous regardons le minimum de la liste ainsi que son indice afin de l'éliminer. On appelle *looser* le nom de la liste éliminée et *namelooser* le candidat en tête de cette liste. Nous retirons *namelooser* de toutes les listes de transfert. Nous transférons ses voix au candidat que *namelooser* a placé en suivant dans sa liste de transfert. Nous regardons si le candidat ayant reçu les voix dépasse ou non le quorum. S'il dépasse le quorum il est élu sinon on recommence cette phase. La dernière condition de la fonction est d'observer si le nombre de candidats qui reste à placer est égal au nombre de sièges libres. Si c'est le cas, les candidats restants sont automatiquement élus.

À noter toutefois que cette méthode ne prend pas en compte la provenance des voix et qu'il serait possible de l'améliorer. En effet, supposons qu'un candidat que l'on nommera A possède 10 voix à redistribuer et, qu'à l'instant t, sa liste de transfert est ['A', 'C', 'B']. On suppose de plus qu'il manque 2 voix à C pour atteindre le *quorum*. Le candidat "A" va donc distribuer les 2 voix manquantes à "C" et non ses 10 voix. Il reste donc 8 voix " en réserve" pour B. Cette façon de réattribuer les voix sera implémentée dans la version plus complète.

Remarque : Nous avons aussi codé le VUTA en C++ avec l'aide d'un camarade du département Information et Système, le code est disponible en annexe. Le prin-

cipe est identique à celui détaillé plus haut. Néanmoins, cela nous a permis de nous familiariser avec un nouveau langage.

4 Le VUTA en Python, implémentation finale

Nous allons, dans cette partie exposé la version finale (proposée par notre tuteur) plus complète que la précédente, tenant compte de toutes les subtilités du VUTA et notamment de l'origine des voix. Par souci de clarté, les variables seront en italiques, les noms de fonctions en gras et les arguments de fonction entre guillemets.

4.1 Quelques préliminaires

Notre fonction VUTA affiche le déroulement du processus, en concluant par la liste des vainqueurs, et ne renvoie rien.

Les entrées de notre fonction VUTA sont :

— "*lesLtransf_*", la donnée des listes-de-transferts qui sera représentée comme un dictionnaire associant la liste de transfert de chaque candidat à la liste des candidats que ce dernier a classé par ordre de préférence.

— "*lesBull_*", la donnée des nombres de bulletins reçus par chacune de ces listes-de-transferts, encore une fois représentée par un dictionnaire associant chaque liste aux nombres de bulletins qu'elle a reçus.

— "*nSieges*", le nombre de sièges à pourvoir

4.2 Fonction VUTA

```
1
2 def VUTA(lesLtransf_, lesBull_, nSieges):
3   lesCand_ = {cand for ltransf in lesLtransf_.values() for cand in
      ltransf}
4     nVotants = sum(lesBull_[nomListrans] for nomListrans in lesLtransf_
      )
5
6     quorum = nVotants / nSieges
7
8   lesLtransf = {candidat: list(listransfert)
9             for candidat, listransfert in lesLtransf_.items()}
10  lesBull = dict(lesBull_)
11  lesCand = set(lesCand_)
12
13  lesElus = list()
14
15  lesVoix = {cand: 0.0 for cand in lesCand_}
16
17  lesDetails = {cand: {nomLtransf: 0.0 for nomLtransf in lesLtransf}
18                  for cand in lesCand}
```

— *lesCand_* est l'ensemble des candidats. Ici nous incluons la possibilité qu'un candidat ne soit en tête d'aucune liste.

— *nVotants* est le nombre total de votes exprimés.

— *quorum* est le nombre de voix à recueillir pour être élu.

— *lesElus* est l'ensemble des élus, qui sera complété au fur et à mesure.

— *lesVoix* est le nombre de voix allouées à chaque candidat en l'état actuel.

— *lesDetails* précise l'information de *lesVoix* en disant, pour chaque candidat, combien des voix reportées sur son nom proviennent des différentes listes.

Etant donné que nous avons besoin de travailler sur des copies des arguments d'entrée qui seront modifiées au cours du programme sans que les arguments eux-mêmes soient modifiés on crée des copies de nos arguments qui se distinguent des arguments d'origine par l'absence de symbole '_' final. (*lesCand*, *lesBull*)

4.3 Phase d'allocation

```
    while True:
        lesPotentiels = {cand: 0.0 for cand in lesCand}
        for nomLtransf in lesLtransf:
            lesPotentiels[lesLtransf[nomLtransf][0]] += lesBull[
nomLtransf]

        lesBesoins = dict()

        for cand in lesCand:
            if lesPotentiels[cand] > 0:
                lesBesoins[cand] = (quorum - lesVoix[cand]) /
lesPotentiels[cand]

            else:
                lesBesoins[cand] = float('inf')
            besoin, premier = min([(besoin, cand) for cand, besoin in
 lesBesoins.items()])
        ecqVidelurne = besoin >= 1

        if ecqVidelurne:
            besoin = 1

        else:

        for nomLtransf in lesLtransf:
            voix = besoin * lesBull[nomLtransf]
            lesDetails[lesLtransf[nomLtransf][0]][nomLtransf] += voix
            lesVoix[lesLtransf[nomLtransf][0]] += voix

            lesBull[nomLtransf] -= voix

        if ecqVidelurne:
            break
        else:

            lesElus.append(premier)
            lesCand.remove(premier)
            del lesDetails[premier]
            del lesVoix[premier]
            for nomLtransf in lesLtransf:
                lesLtransf[nomLtransf].remove(premier)
        if len(lesCand) == 0:

            break
```

La principale différence avec l'algorithme de la partie III est que nous retraçons l'origine des voix et les allouons en conséquence. Pour chaque candidat, après chaque

phase d'allocation, on connait la provenance des voix qui lui sont attribuées grâce au dictionnaire $lesDetails$. Au début de chaque phase d'allocation on résume la situation :

— Candidats encore en course : $lesCand$
— Nombre de voix actuellement obtenues par les différents candidats : $lesVoix$
— Origine des voix des différents candidats : $lesDetails$
— Structure actuelle des listes de transfert : $lesLtransf$
— Nombre de voix qu'il reste à une liste après les jeux d'allocation/élimination : $lesBull$

Ce résumé nous permet d'avoir une idée précise du mode d'allocation des voix pour chaque candidat.

Notre idée est d'utiliser une boucle while de laquelle on ne sortira que s'il ne reste plus de voix à allouer ou qu'il n'y ait plus aucun candidat en jeu. Dans le reste de l'explication on appellera l'urne l'ensemble des voix à allouer. En réalité, dans la première boucle while, on continue d'itérer tant qu'on peut faire de l'**Allocation**, en d'autres termes, avec le report de voix selon le processus de transfert on peut faire élire des candidats sans éliminer de candidat. C'est pour cette raison que l'on teste, à chaque tour de boucle s'il reste des voix dans l'urne.

Dans un premier temps on regarde le candidat dont la liste a besoin du minimum de voix pour atteindre le quorum. Pour ce faire on utilise un dictionnaire $lesBesoins$ qui indique, pour chaque candidat, quelle proportion de l'urne il a besoin qu'on dépouille pour atteindre le quorum. Si le candidat n'a dans l'urne aucun bulletin qui se reporterait sur lui, afin d'éviter les divisions par zéro, on définit son besoin comme étant infini. On veut être sûr de ne perdre aucun bulletin. Ces bulletins que l'on pourrait perdre serait ceux qui sont dépouillés alors qu'un candidat a déja atteint le quorum. On cherche donc à dépouiller uniquement la fraction de tous les bulletins qui se reportent sur un candidat pour qu'il atteigne le quorum. Pour calculer les valeurs associées aux clés du dictionnaire $lesBesoins$, on fait appel au dictionnaire $lesPotentiels$. Ce dictionnaire associe à un candidat l'ensemble des voix qui pourraient potentiellement se reporter vers lui à un instant t du dépouillement. On cherche l'argument du minimum de $lesBesoins$, appelé $premier$, et la valeur correspondante, appelée $besoin$. L'argument correspondra au candidat qui atteindra le quorum en premier. On vérifie que $besoin$ ne dépasse pas 1 car cela reviendrait à dire que l'on peut vider l'urne sans qu'aucun candidat atteigne le quorum. Si ce n'est pas le cas, on alloue les voix restantes aux autres candidats suivant la liste de transfert du candidat élu (l'allocation est fondamentalement différente de ce qui était effectué dans la partie III où on attribuait toutes les voix supplémentaires de « premier » au candidat classé le plus haut sur sa liste de transfert, ici on attribue uniquement ce dont ce dernier a besoin et on alloue le reste des voix aux suivants sur la liste de transfert), tout en stockant leur provenance dans $lesDetails$, on retire tous les bulletins correspondants de l'urne pour ne pas les recompter lors d'une prochaine allocation, et on retire le candidat correspondant à « premier » de tous les objets le concernant. Finalement, on vérifie qu'il reste bien des candidats avant de refaire la boucle **while**.

Dans le cas ou $besoin$ dépasse 1 , c'est-à-dire si en réallouant toutes les voix de l'urne, plus aucun candidat ne peut atteindre le quorum et s'il reste toujours des candidats, on passe à la phase d'élimination.

4.4 Phase d'Élimination

```
        if len(lesCand) <= 1:
            break

        voix, perdant = min([(voix, perdant) for
                             perdant, voix in lesVoix.items()])

        lesBull = lesDetails[perdant]
          lesCand.remove(perdant)
          del lesDetails[perdant]
          del lesVoix[perdant]
          for nomLtransf in lesLtransf:
              lesLtransf[nomLtransf].remove(perdant)
```

Après l'allocation, l'urne est vide. Il va donc falloir éliminer l'un des candidats sauf dans deux situations : s'il n'y a plus aucun candidat, auquel cas il faut arrêter le programme, ou s'il ne reste qu'un candidat en lice. Si nous ne sommes dans aucune des deux situations, on cherche le candidat qui a reçu le moins de voix, appelé *perdant* ,on remet dans l'urne les voix qui s'étaient portées sur le candidat *perdant* grâce au dictionnaire *lesDetails* ,on retire le candidat *perdant* de tout ce qui était susceptible de le concerner, puis on repart sur une phase d'allocation.

Dans le cas où il n'y a plus aucun candidat, il n'y a plus rien à faire. S'il reste un ultime candidat, il faut déterminer s'il doit être élu (si le nombre de voix qu'il a reçu est supérieur au quorum) ou éliminé et procéder à quelques vérifications, détaillées dans la partie code ci-dessous.

```
    if len(lesCand) == 1:
        ultime = list(lesCand)[0]

        for nomLtransf in lesLtransf:
            voix = lesBull[nomLtransf]
            lesDetails[ultime][nomLtransf] += voix
            lesVoix[ultime] += voix
        if lesVoix[ultime] > quorum / 2:
            lesElus.append(ultime)
        else:
        lesCand.remove(ultime)
        del lesVoix[ultime]
        del lesDetails[ultime]
        for nomLtransf in lesLtransf:
            lesLtransf[nomLtransf].remove(ultime)

    return lesElus
```

5 Modélisation

La deuxième partie de notre projet consistait à porter un regard critique sur le VUTA, le but du projet étant de pouvoir dire si le VUTA possède un avantage sur le système de vote classique que l'on connait en France par exemple.

Nous disposons jusque là du programme qui permet de déterminer les vainqueurs d'une éléction suivant le mode de scrutin VUTA. On va vouloir mettre à l'épreuve le VUTA en créant des jeux de données cohérents. C'est la raison pour laquelle nous avons dû effectuer cette étape de modélisation.

5.1 Choix de Modélisation

Nous devons construire à la fois des listes de transfert cohérentes et un ensemble d'électeurs. Pour cela nous avons adopté un modèle s'inspirant de l'idée du marchand de glaces sur la plage.

5.1.1 Le Marchand de Glaces

Supposons que plusieurs marchands de glaces sur une plage représentent les candidats. Un vacancier sera assimilé à electeur. Le vacancier désireux de prendre une glace (voter pour l'électeur) va donc choisir le marchand qui lui est le plus proche. Le schéma suivant représente cela.

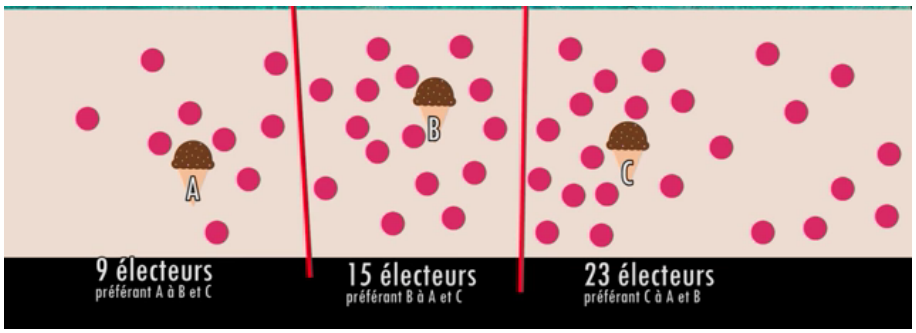

FIGURE 2 – Principe de base du marchand de glace [Wina]

Notre idée est d'utiliser le même principe mais pour une élection.

5.1.2 Modélisation du marchand de glaces

Nous avons trouvé ce modèle très proche de celui d'un choix rationnel pour un électeur. Les candidats se positionnent sur des idées et les électeurs ayant leurs propres idées vont donc déterminer de quel candidat ils sont le plus proches. Pour matérialiser cette notion de distance, il nous a fallu choisir un espace. Nous choix s'est porté sur l'espace $\mathbb{R}^6$. Chaque électeur ainsi que chaque candidat est représenté par un vecteur de $\mathbb{R}^6$. Chaque composante du dit vecteur représente une idée sur un des grands thèmes de la vie politique.Un exemple de vecteur possible est le vecteur ci-dessous.

$$\overrightarrow{Vecteur\_idee} = \begin{pmatrix} Migratoire \\ Interieure \\ Exterieure \\ Economie \\ Environnement \\ Education \end{pmatrix}$$

En représentant comme cela les candidats, il est également possible pour eux de se classer entre eux, en utilisant la notion de distance exposée plus haut.

Ce modèle peut donc permettre de constituer les listes de transferts des candidats mais aussi permet de déterminer les votes des électeurs. Nous expliquerons dans la suite comment on simule ces vecteurs d'idées.

5.1.3 La notoriété d'un candidat

En effet, après avoir testé ce premier modèle, il nous a paru interessant de le compléter pour qu'il se rapproche le plus possible de la réalité. Nous avons toujours gardé notre modèle du marchand de glaces mais, en considérant cette fois ci que certains sont plus connus que d'autres. C'est la notion de **notoriété**.

On peut, à l'instar de ce que nous observons Figure 3 définir la notoriété comme un nouveau paramètre pour le calcul des distances. En effet, comme on le voit sur la FIGURE 3, la notoriété fait grossir un candidat et réduit sa distance aux électeurs.

FIGURE 3 – La notoriété

Les candidats ayant une grande notoriété, attirent alors l'électorat des plus petits candidats. Cette notoriété viendra se retirer au calcul de la distance entre candidat/-candidat et électeur/candidat.

Maintenant que nous avons vu en quoi consistait le principe de la modélisation pour laquelle nous avons optée, nous allons expliquer comment nous avons mis en oeuvre mathématiquement et algorithmiquement ce modèle.

5.2 Le modèle mathématique

5.2.1 Le vecteur d'idée

Comme expliqué précédemment, chaque candidat et chaque électeur possède un vecteur d'idée. Il s'agit donc d'un vecteur de $\mathbb{R}^6$. Il permet de quantifier une idée.

5.2.2 La double gaussienne

Chaque idée politique (donc chaque composante du vecteur) est notée de -10 à 10. -10 correspondant à une idée d'extrême gauche et +10 d'extrême droite. De plus,

ces idées, sont réparties selon une loi de densité :

$$f_X(x) = \frac{1}{2 * \sqrt{2 * \pi * \sigma^2}}(exp(-\frac{x - \mu_1}{2 * \sigma^2}) + exp(-\frac{x - \mu_2}{2 * \sigma^2})) \quad (2)$$

Avec $\mu_1 = -5$ et $\mu_2 = +5$. On aura donc des idées majoritairement de gauche et de droite modérée. C'est d'ailleurs ce que l'on peut observer dans la vie politique Américaine,par exemple.

On peut donc représenter la répartition des idées comme suit :

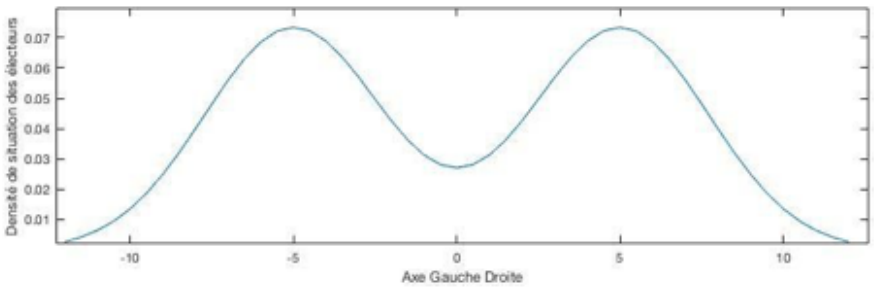

FIGURE 4 – Répartition des idées politiques

Nous avons opté pour une méthode de Box-Muller pour simuler cette loi. Les résultats issus de la simulation sont ceux attendus comme on peut le voir sur la FIGURE 5.

Dans un premier temps, nous avons simulé indépendamment chaque composante du vecteur. Nous avons ensuite considéré que les différentes composantes pouvaient être corrélées. Nous avons donc simulé un vecteur issue de copules afin d'introduire cette idée de dépendance entre les positions politiques.Cette façon de faire sera détaillée un peu plus loin dans le rapport .

Nous avons donc, lors de la phase de simulation sur Python, simulé des vecteurs suivant notre loi "double gaussienne", de variance $\sigma^2 = 2$ et avec ($\mu_1 = -5$, $\mu_2 = 5$).

5.2.3 Calcul des distances

Pour former les listes de transfert ainsi que pour rentrer dans le programme VUTA_v90401 un résultat de dépouillement cohérent, il faut effectuer des calculs de distances. Pour cela nous utilisons la norme euclidienne définie par :

$$d(Idee1, Idee2) = \sum_{i=0}^{n}(Idee1[i] - Idee2[i])^2 \quad (3)$$

5.2.4 Construction des listes de transfert

Un candidat, pour constituer sa liste de transfert, va calculer sa distance avec tous les autres candidats et va les classer par distances croissantes. Ainsi, on aura des listes de transfert cohérentes respectant bien notre modèle.

5.2.5 Choix d'un électeur

Fort des choix mathématiques précédents, nous pouvons déterminer pour quelle liste de transfert un électeur va voter. Pour cela, dans un premier temps, l'électeur va observer sa distance avec tous les candidats qui se présentent.

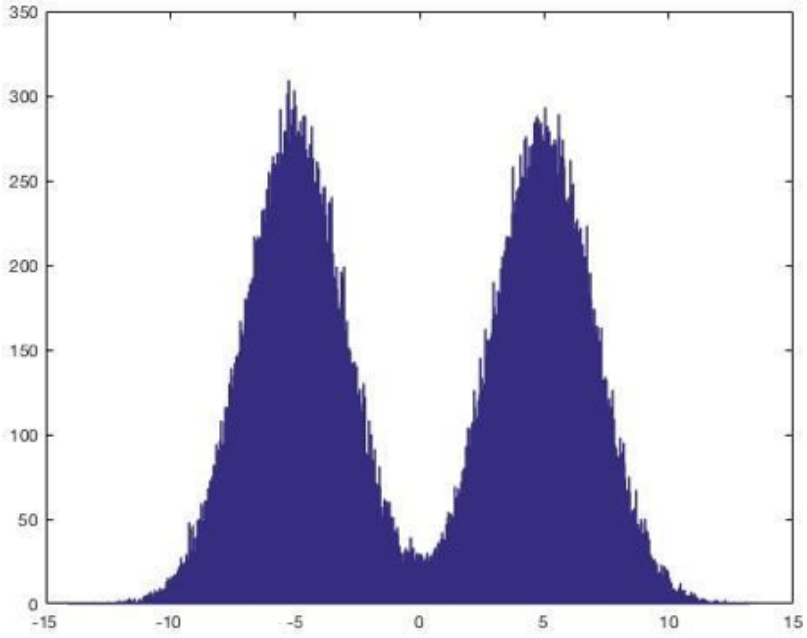

FIGURE 5 – Histogramme issue de la simulation informatique

Ensuite, pour choisir la liste pour laquelle il va voter, il va sommer avec des coefficients de pondération, les distances qui le séparent des 10 premiers candidats des listes . Cette pondération est introduite pour tenir compte du fait qu'un électeur est plus concerné par les candidats situés dans les premières positions de la liste. Ainsi, un électeur va chercher à minimiser sa distance à une liste. Il va donc satisfaire les équations suivantes :

$$\min_{liste \in listes\_de\_transfert} \sum_{k=1}^{10} \frac{1}{k} * d(electeur, liste[k]) \tag{4}$$

Dans le cas où l'on introduit la notoriété définie précédemment, on aura :

$$\min_{liste \in listes\_de\_transfert} \sum_{k=1}^{10} \frac{1}{k} (d(electeur, liste[k]) - notoriete[candidat\_en\_k^{i-eme} position]) \tag{5}$$

On peut donc déterminer grâce à ce procédé les votes de tous les électeurs.

5.3 Simulation à l'aide de copules

5.3.1 Utilité

Les copules sont un objet permettant d'identifier et de quantifier la force de dépendance qui existe entre les différentes coordonnées d'une variable aléatoire à valeurs dans $\mathbb{R}^d$ sans se préoccuper de ses lois marginales. Les variables aléatoires dans notre cas étant les vecteurs d'idée, notre objectif est de simuler un vecteur d'idées cohérent. En effet, jusqu'à présent nous simulions les composantes du dit vecteur de manière aléatoire mais en réalité il serait absurde de ne pas tenir compte des corrélations entre ses différentes composantes. Par exemple, la politique migratoire et la politique intérieure étant liées, un candidat de droite ne peut avoir des

composantes opposées sur l'échiquier politique dans son vecteur d'idées. On cherche donc à simuler des vecteurs d'idées dont certaines composantes seraient corrélées.

5.3.2 Définition

[PLA18]

Par souci de concision, nous avons choisi de ne détailler que brièvement la théorie mathématique.

Soit $(X_1, X_2, \cdots, X_d)$ un vecteur aléatoire.
Supposons que les fonctions de répartition des variables aléatoires $X_1, X_2, \cdots, X_d$ respectivement $F_1, F_2, \cdots, F_d$ soient continues.
Soit $(U_1, U_2, \cdots, U_d) = (F_1(X_1), F_2(X_2), \cdots, F_d(X_d))$.
$\forall i \in [\![1, d]\!]$, la variable $U_i = F_i(X_i)$ est distribuée suivant une loi uniforme sur $[0, 1]$. En effet, $\mathbb{P}(U_i \leqslant u_i) = \mathbb{P}(X_i \leqslant F_i^{-1}(u_i)) = u_i$. La copule associée au vecteur aléatoire $(X_1, X_2, \cdots, X_d)$ notée C est la fonction de répartition multivariée de $(U_1, U_2, \cdots, U_d)$. En d'autres termes, $\mathrm{C}(u_1, u_2, \cdots, u_d) = \mathbb{P}(U_1 \leqslant u_1,\ U_2 \leqslant u_2,\ \cdots,\ U_d \leqslant u_d)$. L'interêt principal des copules réside dans le fait qu'il est possible de simuler une variable aléatoire multivariée à partir de sa copule et de ses lois marginales. Il suffit pour cela de générer un vecteur aléatoire $(U_1, U_2, \cdots, U_d)$ à partir de la copule et de construire l'échantillon voulu grâce à la relation : $(X_1, X_2, \cdots, X_d) = (F_1^{-1}(U_1), F_2^{-1}(U_2), \cdots, F_d^{-1}(U_d))$.
A noter toutefois que ceci n'est possible que parce que nous avons supposé que les fonctions de répartitions étaitent inversibles, si ce n'était pas le cas, il aurait fallu définir l'inverse généralisée. Nous n'approfondirons pas ce point car seule la simulation numérique du vecteur$(X_1, X_2, \cdots, X_d)$ nous intéresse et qu'il est toujours possible d'approcher l'inverse d'une fonction numériquement (dichotomie, Newton, etc..).

5.3.3 Simulation d'un vecteur aléatoire à composantes corrélées

Soit C1, C2, C3, C4, C5, C6 les composantes des vecteurs d'idées, associées respectivement à : la politique migratoire, intérieure, extérieure, économique, environnementale, de l'éducation.
On définit ensuite une matrice de covariance ayant des coefficient attribués proportionnellement à la proximité des différents axes politiques. Nous avons choisi de corréler ces composantes suivant le tableau ci-dessous. Les coefficients de corrélations sont des paramètres de notre programme et sont donc modifiables, le tableau ci-dessous est donné à titre d'exemple.

	C1	C2	C3	C4	C5	C6
C1	1	0.9	0.9	0	0	0
C2	0.9	1	0.8	0	0	0
C3	0.8	0.9	1	0	0	0
C4	0	0	0	1	0	0.7
C5	0	0	0	0	1	0
C6	0	0	0	0.7	0	1

FIGURE 6 – Matrice de corrélation entre idées politiques

En appliquant le procédé presenté dans la partie précédente, on simule un vecteur gaussien $(X_1, X_2, \cdots, X_6)$ de matrice de covariance C ayant pour coefficient ceux du tableau ci-dessus. On calcule ensuite $(X_{1b}, X_{2b}, \cdots, X_{6b}) = (F(X_1), F(X_2), \cdots, F(X_6))$ où F est la fonction de répartition de la loi normale. Finalement, le vecteur d'idées avec composantes corrélées, noté $(Y_1, Y_2, \cdots, Y_6)$, est obtenu par l'opération $(Y_1, Y_2, \cdots, Y_6) = (F_{db}^{-1}(X_{1b}), F_{db}^{-1}(X_{2b}), \cdots, F_{db}^{-1}(X_{6b}))$ où F_{db}^{-1} est l'inverse de notre fonction de répartition en "double bosse". Cet inverse a été approché numériquement grâce à la méthode de dichotomie. Pour plus de détails, se référer au code en annexe.

Par ailleurs, on peut tracer un scatter plot pour observer la répartition de chaque idée sur l'échelle -10,10 ainsi que les positions issues de la simulation des copules.

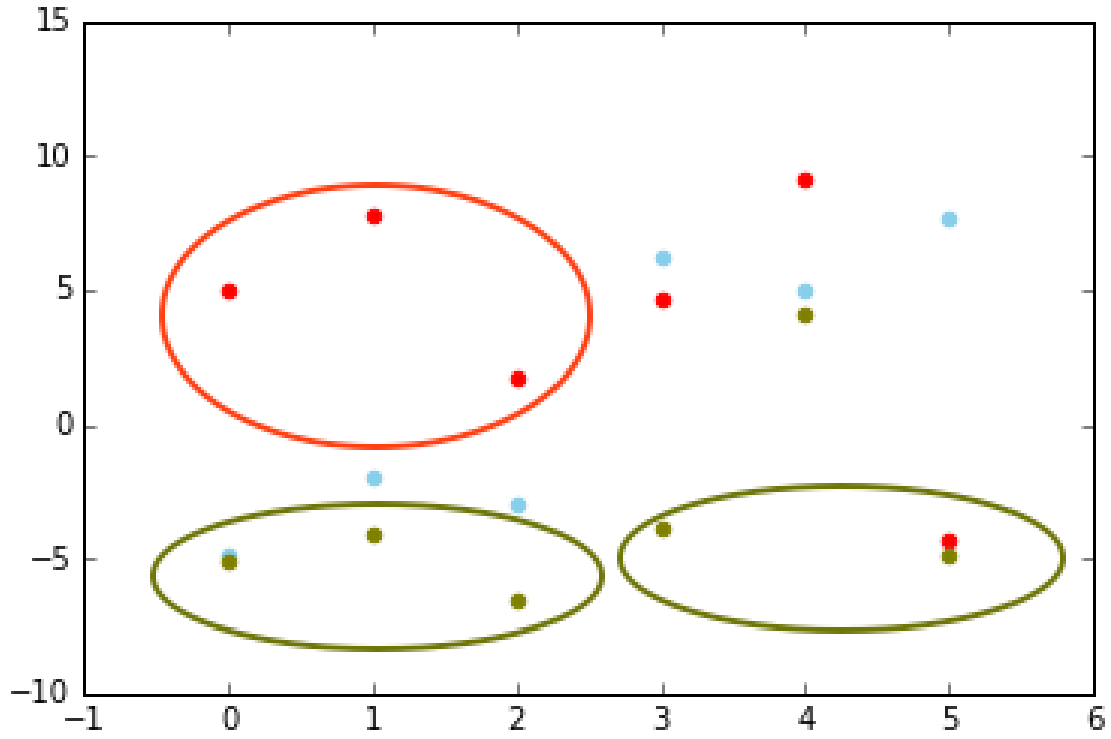

FIGURE 7 – Simulation des idées en utilisant des copules Gaussiennes. Tous les points de même couleurs sont issues du même vecteur.

On observe bien la corrélation entre les points 1,2,3 ainsi qu'entre 4 et 6.

5.4 Validation du mode de scrutin VUTA

Afin de valider ce mode de scrutin, il nous faut tester nos jeux de données obtenus avec la simulation. Pour cela, nous allons organiser virtuellement deux types de scrutins sur des lois. Dans un premier temps, nous allons soumettre aux électeurs une loi. Ils devront, selon un critère que nous allons expliquer dans la partie suivante, voter en faveur ou non de cette loi. Ensuite, nous soumettrons cette même loi aux élus, qui ont été élus selon le VUTA et ceux élus suivant le mode de scrutin classique.

Le but ensuite, est d'observer les résultats. Le mode de scrutin qui semblera le plus adapté sera alors celui où le pourcentage de oui des élus (resp. non) sera le plus proche du pourcentage de oui (resp. non) des électeurs.

5.5 Vote sur une Loi

Une loi, est à nouveau un vecteur de $\mathbb{R}^6$ avec les mêmes composantes. Un électeur ou un candidat, va donc calculer sa distance à la loi et, si la distance est inférieure à une variable $SEUIL$ que l'on fixe (en tenant compte de l'ordre de grandeur des distances) de manière à avoir des résultats réalistes et analysables, il votera pour oui. Sinon ce sera non. C'est ainsi en testant notre programme que nous pourrons conclure sur le bien fondé du VUTA par rapport à d'autres modes de scrutin.

6 Résultats

Nous avons déterminé le pourcentage de "oui" des élus ainsi que des électeurs pour 100 lois différentes en utilisant les modèles définis plus haut : sans notoriété, avec notoriété , avec et sans copules et nous comparons les résultats avec un mode de scrutin classique i.e. sans listes de transfert.

Dans un premier temps, nous avons testé les modèles avec 1000 candidats en lice pour 350 places et 10^6 élécteurs. Pour des raisons de puissance de calculs pour ces valeurs de paramètres, nous n'obtenons pas de résultas pour la simulation avec copules. Néanmoins au bout de 6h environ (plus précisément 23540 secondes), nous obtenons les résultats présentés dans la FIGURE 8 pour les deux modèles sans copules.

```
Le Pourcentage de Oui des electeurs est de 0.292127
Le Pourcentage de Oui des candidats sans Notoriété est de 0.294029
Le Pourcentage de Oui des candidats avec Notoriété est de 0.290571
Le Pourcentage de Oui des candidats sans Listes de Transfert est de 0.290743
```

FIGURE 8 – Résultats de la simulation informatique sans copules

Afin de déterminer le modèle le plus proche du choix des électeurs nous calculons la distance relative entre les pourcentages de "oui" de chaque modèle et celui des électeurs. Les résultats sont récapitulés dans le tableau ci-dessous.

Modèle	Sans Notoriété	Avec Notoriété	Classique
Distance relative par rapport aux électeurs	0.001902	0.001556	0.001384

FIGURE 9 – Distances relatives par rapport aux élécteurs sans copules

Ce tableau ne nous permet pas de conlure car les différences entre les distances sont trop faibles. Ces deux modèles ne nous permettent pas de trancher, c'est d'ailleurs la raison pour laquelle nous avons introduit un modèle plus réaliste tenant compte de la corrélation entre idées.

Du fait du temps de calcul important requis pour la simulation avec copules, nous avons réduit le nombre d'électeurs à 100000 et le nombre de candidats à 50 pour 25 places.

Les résultats obtenus sont les suivants :

```
Le Pourcentage de Oui des electeurs est de 0.102320
Le Pourcentage de Oui des candidats sans Notoriété est de 0.095600
Le Pourcentage de Oui des candidats avec Notoriété est de 0.102400
Le Pourcentage de Oui des candidats sans Listes de Transfert est de
0.120000
```

FIGURE 10 – Résultats de la simulation informatique avec copules

Nous récapitulons les distances relatives par rapport aux électeurs dans le tableau ci-dessous :

Modèle	Sans Notoriété	Avec Notoriété	Classique
Distance relative par rapport aux électeurs	0.00672	0.00008	0.001768

FIGURE 11 – Distances relatives par rapport aux électeurs avec copules

Avec cette simulation, il semblerait que le modèle avec notoriété en utilisant le scrutin VUTA est le plus représentatif des élécteurs. Réitérer cette simulation un grand nombre de fois avec des conditions similaires à la première nous permettrait de conclure. Toutefois le temps de calcul nous ayant bloqué nous ne pouvons valider avec certitude cette première conjecture qui semble pencher en faveur du VUTA.

7 Conclusion

En conclusion, à l'heure actuelle en raison de nos limitations en terme de puissance de calcul, nous n'avons pas pu trancher de manière claire sur l'intérêt pratique du VUTA, malgré son aspect conceptuel intéressant.Par ailleurs, ce projet nous a été bénéfique dans le sens où il nous a fait appliquer nos connaissances théoriques en informatique et mathématiques en les inscrivant dans une démarche d'ingénieur. Afin de répondre à la problématique de l'intérêt ou non du VUTA, il nous a fallu dans un premier temps s'approprier le principe théorique qui nous a été expliqué par notre tuteur, l'implémenter dans un langage de programmation adéquat et enfin, faire appel à nos capacités d'analyse et de modélisation mathématique afin de simuler des situations proches de la réalité, nous permettant de tester le VUTA dans des conditions réalistes. Ces différentes étapes de travail nous ont permis de mieux appréhender comment un ingénieur doit traiter une problématique industrielle. Nous pensons avoir progressé à la fois dans la gestion de projet, mais aussi dans la manière de partager nos informations avec les personnes gravitants autour de ce projet ce qui pour nous sont des éléments essentiels dans la réussite d'une carrière d'ingénieur. Ce projet est donc dans ce sens, une étape clé dans notre formation.

8 Annexes Codes Python

Les Codes sont disponibles à l'adresse dropbox suivante :
"https ://www.dropbox.com/sh/2davbsr4qg3q5a3/AABlirfrrHaTxVYuEYLZ90EZa ?dl=0"

8.1 Première version du VUTA en Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#VUTA

import random #utile pour definir la liste des votants
import numpy as np # pour pouvoir manipuler des structures matricielles

## FONCTION AUXLIAIRES

# "def_dic_tete_de_liste" crée un dictionnaire qui, à un nom de liste-de-transferts, associe le nom du candidat en tête de cette liste. L'argument "LT" est la liste des listes_de_candidats-de-transferts, et l'argument "noms_des_listes_de_candidats" est la liste des noms respectifs de ces listes_de_candidats-de-transferts.
def def_dic_tete_de_liste(LT,noms_des_listes_de_candidats):
    d = dict()
    for k in range(len(noms_des_listes_de_candidats)):
        d[noms_des_listes_de_candidats[k]] = LT[k][0]
    return d

# "def_dic_indice_candidat" crée un dictionnaire qui, à un nom de liste-de-transferts, associe un numéro l'identifiant. L'argument "noms_des_listes_de_candidats" est la liste des noms de listes_de_candidats-de-transferts.
def def_dic_candidat_indice(noms_des_listes_de_candidats):
    d = dict()
    for k in range(len(noms_des_listes_de_candidats)):
        d[noms_des_listes_de_candidats[k]] = k
    return d



# "invertdict" produit le dictionnaire réciproque d'un dictionnaire "dictionnaire" dont toutes les valeurs sont distinctes, les valeurs devenant clés et vice-versa.
def invertdic(dictionnaire):
    return dict([(v,k) for k,v in dictionnaire.items()])

# Cette fonction réalise le dépouillement elle renvoie le nombre de voix associées à chaque liste
def comptage_list(choix_electeurs,dic_indice_candidat):
    Lcompt = [0 for k in range(len(LT))] # crée une liste avec autant d'entrées que de listes_de_candidats-de-transferts, initialisée à zéro.
    for k in choix_electeurs:
        Lcompt[dic_indice_candidat[k]]+=1
    return np.array(Lcompt)


```

```
39 # comptage_pers prend en entrée le liste "LT" des listes_de_candidats-
       de-transferts, et la liste "resultat_allocation_liste_candidats"
       des nombres de voix respectifs obtenus par chacun de ces
       noms_des_listes_de_candidats ; et renvoie la liste des nombres de
       voix que cela représente pour chaque candidat, les voix pour une
       liste étant allouées au candidat qui en est en tête.
40 def comptage_pers(LT,resultat_allocation_liste_candidats): #utile pour
       faire évoluer
41     n = len(LT)
42     Lc = [0 for k in range(n_candidat)] # "n_candidats" est une
       variable externe qui décrit le nombre de candidats.
43     if len(LT[0])!=0:
44         for k in range(n):
45             v = LT[k][0] # Candidat en tête de la liste-de-transferts
       numéro k.
46             liste = dic_tete_de_liste_inv[v] # "dic_tete_de_liste_inv"
       est le dictionnaire qui à un candidat associe le nom de la liste
       dont il est en tête.
47             rang = dic_candidat_indice[liste] # "rang" est donc le numé
       ro associé au candidat v.
48             Lc[rang] = Lc[rang] + resultat_allocation_liste_candidats[k
       ]
49     return Lc
50
51 # Retourne le minimum des valeurs non nulles de la liste "Liste" et son
       indice (sous la forme d'un couple).
52 def minDiffZero(Liste):
53     mini = max(Liste)
54     indmin = 0
55     for k in range(len(Liste)):
56         if Liste[k] < mini and Liste[k] !=0:
57             mini,indmin = Liste[k],k
58     return (mini,indmin)
59
60 ## FONCTION "VUTA". Les entrées sont "LT", la liste des liste-de-
       transferts, "resultat_allocation_liste_candidats", la liste des
       nombres de voix reçues par chacune de ces listes_de_candidats-de-
       transferts, et "n_sieges", le nombre de sièges à pourvoir et quorum
       , le nombr de voix nécessaires à l'élection. La fonction renvoie
       une liste formée de la chaine 'les vainqueurs sont :' et de la
       liste des vainqueurs.
61 def VUTA(LT,resultat_allocation_liste_candidats,n_sieges,quorum):
62     liste_vainqueurs = list() # "liste_vainqueurs" contiendra la liste
       des vainqueurs, qui sera complétée au fur et à mesure
63     nombre_de_voix_actualisées = resultat_allocation_liste_candidats #
       "nombre_de_voix_actualisées", qui sera actualisé au cours du
       scrutin, est la liste contenant, pour chaque candidat, le nombre de
        voix qui restent à dépouiller et qui  se porteraient vers lui en l
       'état actuel des listes_de_candidats-de-transferts.
64     if len(LT)==n_sieges:
65         liste_vainqueurs = list(noms_des_listes_de_candidats)
66         return liste_vainqueurs# on a pas besoin de passer à la phase d
       'élimination si après la premi    ère
67     # PREMIÉRE PHASE DU DÉPOUILLEMENT: ALLOCATION
68     else:
69         while True:
70             maxi,indmax = np.amax(nombre_de_voix_actualisées),np.argmax
       (nombre_de_voix_actualisées) # Nombre de voix et numéro du candidat
```

```
 qui a le plus de voix.
71          print("le plus grand nombre de voix obtenu par une liste
    electorale est :  ", maxi)
72          if maxi>=quorum:
73              winner = dic_indice_candidat[indmax]
74              print("le vainqueur est donc :",winner)
75              liste_vainqueurs.append(winner)
76              toquorum = float(maxi-quorum) # Nombre de voix en plus
    du quorum reçues par le liste_vainqueurs.
77              for k in range(len(LT)): # Maintenant on retire de "
    resultat_allocation_liste_candidats" les bulletins qui ont servi à
    faire élire le liste_vainqueurs.
78                  if LT[k][0] == dic_tete_de_liste[winner]:
79                      resultat_allocation_liste_candidats[k] =
    resultat_allocation_liste_candidats[k]/float(maxi)*float(toquorum)
80              # On retire le candidat élu de la liste
    noms_des_listes_de_candidats
81              LT = [[element for element in row if element !=
    dic_tete_de_liste[dic_indice_candidat[indmax]]] for row in LT]
82              # Il faut remettre à jour le nombre de voix pour chaque
     individu, puisqu'on a retiré certains bulletins.
83              print("resultat de l'allocation de voix : " ,
    resultat_allocation_liste_candidats)
84              print("LT : ",LT)
85              nombre_de_voix_actualisées = comptage_pers(LT,
    resultat_allocation_liste_candidats)
86              print("nombre_de_voix_actualisées :",
    nombre_de_voix_actualisées)
87              print("Liste de Transfert:",LT)
88
89
90          else: # Si aucun candidat n'a un nombre suffisant de voix
    pour atteindre la quorum, il faut passer à la phase d'élimination.
91              break
92
93      # SECONDE PHASE DU DÉPOUILLEMENT: ÉLIMINATION
94      print("phase d'élimination")
95      while True:
96          mini,indmini = minDiffZero(nombre_de_voix_actualisées)[0],
    minDiffZero(nombre_de_voix_actualisées)[1] # "indmini" est le numé
    ro du candidat ayant reçu le moins de voix (sauf nombre de voix nul
    ), et "mini" le nombre de voix correspondant.
97
98          looser = dic_indice_candidat[indmini] # "looser" est le nom
     de la liste du candidat ayant reçu le moins de voix.
99          namelooser = dic_tete_de_liste[looser] # et "namelooser"
    est finalement le nom en lui-même du candidat ayant reçu le moins
    de voix.
100
101         print("le candidat éliminé est : " , namelooser)
102
103         LT = [[element for element in row if element != namelooser]
     for row in LT] # On retire le candidat perdant de toutes les
    listes_de_candidats-de-transferts.
104         # On va maintenant transférer les voix reçues par le
    candidat éliminé au candidat situé le plus haut sur sa liste de pré
    férences.
105
```

```
106             transfert = LT[indmini][0] # Nom du candidat qui recevra
    les voix du candidat éliminé.
107             indtransfert = dic_candidat_indice[dic_tete_de_liste_inv[
    transfert]] # Numéro de ce candidat
108
109
110
111             nombre_de_voix_actualisées[indtransfert] =
    nombre_de_voix_actualisées[indtransfert] + nombre_de_voix_actualisé
    es[indmini] # On donne les voix du candidat éliminé à son second
    choix.
112             print(nombre_de_voix_actualisées[indtransfert])
113             toquorum = nombre_de_voix_actualisées[indtransfert]-quorum
114
115             nombre_de_voix_actualisées[indmini] = 0 # On jette donc les
     voix du candidat éliminé
116             print("le nombre de voix en trop par rapport au quorum est
     :",toquorum)
117
118             if toquorum >=0: # Si il y a un surplus de voix, un
    candidat est élu mais on ne lui attribue que ce dont il a besoin
119                 liste_vainqueurs.append(dic_tete_de_liste_inv[transfert
    ])
120                 LT = [[element for element in row if element !=
    transfert] for row in LT] # On supprime l'individu qui vient d'être
     élu des listes_de_candidats-de-transferts.
121                 nombre_de_voix_actualisées[indtransfert] = 0
122
123
124
125             if len(LT[0]) == n_sieges - len(liste_vainqueurs):
126                 break
127
128
129         for k in LT[0]: # Si il reste autant de places que de candidats
     ils sont tous élus
130             print("Il reste autant de places que de candidats restants"
    )
131             liste_vainqueurs.append(dic_tete_de_liste_inv[k])
132     return ('les vainqueurs sont :', liste_vainqueurs)
133
134
135 # PROGRAMME PRINCIPAL
136 if __name__ =='__main__':
137     # listes_de_candidats-de-transferts données par les candidats.
138     LA = ["A","B","C","E","D"]
139     LB = ["B","E","D","A","C"]
140     LC = ["C","D","A","B","E"]
141     LD = ["D","B","E","A","C"]
142     LE = ["E","A","B","D","C"]
143
144     LT = [LA,LB,LC,LD,LE]
145
146     noms_des_listes_de_candidats = ("LA","LB","LC","LD","LE") # Noms
    des listes_de_candidats.
147
148     n_candidat = len(LT) # Nombre de candidats
149     print ('il y a', n_candidat, 'candidats');
```

```
150     dic_candidat_indice = def_dic_candidat_indice(
        noms_des_listes_de_candidats) # Dictionnaire atribuant un numéro à
        chaque nom de liste-de-transfert.
151     dic_tete_de_liste = def_dic_tete_de_liste(LT,
        noms_des_listes_de_candidats) # Dictionnaire qui à chaque nom de
        liste-de-transfert, associe le nom du candidat en tête de cette
        liste.
152
153
154     dic_indice_candidat = invertdic(dic_candidat_indice) # "
        dic_candidat_indice" associe à chaque numéro le nom de la liste-de-
        transferts correspondante.
155     dic_tete_de_liste_inv = invertdic(dic_tete_de_liste) # Dictionnaire
         qui, à chaque nom de candidat, associe le nom de la liste-de-
        transfert dont ce candidat est en tête.
156
157     nvotants = 150
158     n_sieges = 4
159
160     print ('il y a', n_sieges, 'sièges à pourvoir');
161
162     quorum = nvotants/n_sieges
163
164     resultat_allocation_liste_candidats = np.array([77,29,13,14,17],
        dtype = np.float) # Nombre de voix pour chaque liste.
165     vainqueurs = VUTA(LT,resultat_allocation_liste_candidats,n_sieges,
        quorum)
166     print(vainqueurs)
```

8.2 Première version du VUTA implémentée en C++

```
1  // VUTAsansDicorigine.cpp : Defines the entry point for the console
       application.
2  //
3
4  #include "stdafx.h"
5
6  #include <map>
7  #include <vector>
8  #include <string>
9  #include <iostream>
10
11 #include "fonctionsCommunes.h"
12
13 using namespace std;
14
15 /**
16 * Fonction principale qui simule une election
17 * @param LT la liste des listes de transfert
18 * @param n_sieges le nombre de sieges a pourvoir
19 * @param quorum le nombre de voix necessaire pour être elu
20 * @return la liste des vainqueurs
21 */
22 vector<string> VUTAsansDicorigine(vector<vector<string>*>& LT, vector<
       float> resultat_allocation_liste_candidats, int n_sieges, float
       quorum,
23   map<int, string>& dic_indice_candidat, map<string, string>
       dic_tete_de_liste, map<string, string>& dic_tete_de_liste_inv, map<
```

```
    string, int>& dic_candidat_indice,
24  int  n_candidats)
25 {
26  vector<string>* listVainqueurs = new vector<string>();
27  vector<float> nombre_de_voix_actualise =
     resultat_allocation_liste_candidats;
28
29  float maxi(0), toquorum(0); int indmax(-1); string winner;
30
31  ///PREMIÉRE PHASE DU DÉPOUILLEMENT : ALLOCATION
32
33  while (true) {
34    maxi = 0; indmax = -1;
35    for (int k = 0; k < nombre_de_voix_actualise.size(); ++k) {
36      if (nombre_de_voix_actualise[k] > maxi) {
37        maxi = nombre_de_voix_actualise[k];
38        indmax = k;
39      }
40    }
41
42    cout << "le plus grand nombre de voix obtenu par une liste
     electorale est : " << maxi << endl;
43
44    if (maxi >= static_cast<float>(quorum)) {
45      winner = dic_indice_candidat[indmax];
46      cout << "Le vainqueur est donc :" << winner << endl;
47      listVainqueurs->push_back(winner);
48      toquorum = maxi - static_cast<float>(quorum); //Nombre de voix en
     plus du quorum reçues par le liste_vainqueurs
49
50      for (int k = 0; k < LT.size(); ++k) {
51        if ((*LT[k])[0] == dic_tete_de_liste[winner]) {
52          resultat_allocation_liste_candidats[k] =
     resultat_allocation_liste_candidats[k] / maxi*static_cast<float>(
     toquorum);
53        }
54      }
55
56      //On retire le candidat elu de la liste
     noms_des_listes_de_candidats
57      for (auto liste : LT)
58      {
59        auto x = find(liste->begin(), liste->end(), dic_tete_de_liste[
     dic_indice_candidat[indmax]]);
60        if (x != liste->end()) liste->erase(x);
61      }
62
63      //Il faut remettre a jour le nombre de voix pour chaque individu,
     puisqu'on a retire certains bulletins
64      cout << "resultat de l'allocation de voix : " <<
     resultat_allocation_liste_candidats << endl;
65      cout << "LT : " << LT << endl;
66      nombre_de_voix_actualise = comptage_pers(LT,
     resultat_allocation_liste_candidats, dic_tete_de_liste_inv,
     dic_candidat_indice, n_candidats);
67      cout << "nombre de voix actualise : " << nombre_de_voix_actualise
     ; cout << endl;
68      cout << "Liste de transfert : " << LT; cout << endl;
```

```
69      }
70      else {//Si aucun candidat n'a un nombre suffisant de voix pour
        atteindre la quorum, il faut passer a la phase d'elimination
71        break;
72      }
73
74    }
75
76    /// SECONDE PHASE DU DÉPOUILLEMENT : ÉLIMINATION
77    cout << "phase d'elimination" << endl;
78
79    float mini;int indmini,indtransfert; string looser, namelooser,
        transfert;
80
81    while (true) {
82      pair<float, int> miniindmini = minDiffZero(nombre_de_voix_actualise
        );
83      mini = miniindmini.first; indmini = miniindmini.second;
84
85      looser = dic_indice_candidat[indmini];
86      namelooser = dic_tete_de_liste[looser];
87
88      cout << "Le candidat elimine est : " << namelooser << endl;
89
90      //On retire le candidat perdant de toutes les listes_electorales de
        candidat_transferts
91      for (auto liste : LT)
92      {
93        auto x = find(liste->begin(), liste->end(), namelooser);
94        if (x != liste->end()) liste->erase(x);
95      }
96
97      transfert = (*LT[indmini])[0];
98      indtransfert = dic_candidat_indice[dic_tete_de_liste_inv[transfert
        ]];
99
100     nombre_de_voix_actualise[indtransfert] = nombre_de_voix_actualise[
        indtransfert] + nombre_de_voix_actualise[indmini];
101
102     cout << nombre_de_voix_actualise[indtransfert] << endl;
103
104     toquorum = nombre_de_voix_actualise[indtransfert] - quorum;
105
106     nombre_de_voix_actualise[indmini] = 0;
107
108     cout << "Le nombre de voix en trop par rapport au quorum est :" <<
        toquorum << endl;
109
110       if (toquorum >= 0) {
111
112         listVainqueurs->push_back(dic_tete_de_liste_inv[transfert]);
113         //On supprime le candidat elu des listes
114         for (auto liste : LT)
115         {
116           auto x = find(liste->begin(), liste->end(), transfert);
117           if (x != liste->end()) liste->erase(x);
118         }
119         nombre_de_voix_actualise[indtransfert] = 0;
```

```
120         }
121
122
123       if (LT[0]->size() == n_sieges - listVainqueurs->size()) break;
124     }
125     for (auto k : *LT[0]) {
126       cout << "Il reste autant de places que de candidats restants";
127       listVainqueurs->push_back(dic_tete_de_liste_inv[k]);
128     }
129
130     return *listVainqueurs;
131 }
132
133 int main()
134 {
135     vector<string> LA, LB, LC, LD, LE;
136     LA.push_back("A"); LA.push_back("B"); LA.push_back("C"); LA.push_back
        ("E"); LA.push_back("D");
137     LB.push_back("B"); LB.push_back("E"); LB.push_back("D"); LB.push_back
        ("A"); LB.push_back("C");
138     LC.push_back("C"); LC.push_back("D"); LC.push_back("A"); LC.push_back
        ("B"); LC.push_back("E");
139     LD.push_back("D"); LD.push_back("B"); LD.push_back("E"); LD.push_back
        ("A"); LD.push_back("C");
140     LE.push_back("E"); LE.push_back("A"); LE.push_back("B"); LE.push_back
        ("D"); LE.push_back("C");
141
142
143     vector<vector<string>*> LT;
144     LT.push_back(&LA); LT.push_back(&LB); LT.push_back(&LC); LT.push_back
        (&LD); LT.push_back(&LE);
145     cout << LT << endl;
146     vector<string> noms_des_listes_de_candidats;
147     noms_des_listes_de_candidats.push_back("LA");
        noms_des_listes_de_candidats.push_back("LB");
        noms_des_listes_de_candidats.push_back("LC");
148     noms_des_listes_de_candidats.push_back("LD");
        noms_des_listes_de_candidats.push_back("LE");
149
150     int n_candidats = LT.size();
151     cout << "il y a " << n_candidats << " candidats" << endl;
152
153     map<string, int>* dic_candidat_indice = def_dic_candidat_indice(
        noms_des_listes_de_candidats);
154     map<string, string>* dic_tete_de_liste = def_dic_tete_de_liste(LT,
        noms_des_listes_de_candidats);
155
156     map<int, string>* dic_indice_candidat = invertdic(*
        dic_candidat_indice);
157     map<string, string>* dic_tete_de_liste_inv = invertdic(*
        dic_tete_de_liste);
158
159     int nvotants = 150;
160     int n_sieges = 4;
161
162     cout << "il y a " << n_sieges << " sieges a pourvoir" << endl;
163
164     float quorum = (float)nvotants / (float)n_sieges;
```

```
165
166   vector<float> resultat_allocation_liste_candidats;
167   resultat_allocation_liste_candidats.push_back(77.0);
168   resultat_allocation_liste_candidats.push_back(29.0);
169   resultat_allocation_liste_candidats.push_back(13.0);
170   resultat_allocation_liste_candidats.push_back(14.0);
171   resultat_allocation_liste_candidats.push_back(17.0);
172
173   vector<string> vainqueurs = VUTAsansDicorigine(LT,
       resultat_allocation_liste_candidats, n_sieges, quorum,
174      *dic_indice_candidat, *dic_tete_de_liste, *dic_tete_de_liste_inv, *
       dic_candidat_indice, n_candidats);
175
176   cout << "Les vainqueurs sont :" << vainqueurs;
177   cout << endl;
178
179   delete dic_candidat_indice;
180   delete dic_indice_candidat;
181   delete dic_tete_de_liste;
182   delete dic_tete_de_liste_inv;
183
184   cin.get();
185   return 0;
186 }
```

8.3 Version finale du VUTA

```
1
2 def VUTA(lesLtransf_, lesBull_, nSieges):
3   lesCand_ = {cand for ltransf in lesLtransf_.values() for cand in
      ltransf}
4     nVotants = sum(lesBull_[nomListrans] for nomListrans in lesLtransf_
      )
5
6     quorum = nVotants / nSieges
7
8   lesLtransf = {candidat: list(listransfert)
9             for candidat, listransfert in lesLtransf_.items()}
10  lesBull = dict(lesBull_)
11  lesCand = set(lesCand_)
12
13  lesElus = list()
14
15  lesVoix = {cand: 0.0 for cand in lesCand_}
16
17  lesDetails = {cand: {nomLtransf: 0.0 for nomLtransf in lesLtransf}
18                  for cand in lesCand}
19
20  while True:
21      while True:
22          lesPotentiels = {cand: 0.0 for cand in lesCand}
23          for nomLtransf in lesLtransf:
24              lesPotentiels[lesLtransf[nomLtransf][0]] += lesBull[
      nomLtransf]
25
26          lesBesoins = dict()
27
28          for cand in lesCand:
```

```
29              if lesPotentiels[cand] > 0:
30                  lesBesoins[cand] = (quorum - lesVoix[cand]) /
    lesPotentiels[cand]
31
32              else:
33                  lesBesoins[cand] = float('inf')
34              besoin, premier = min([(besoin, cand) for cand, besoin in
     lesBesoins.items()])
35          ecqVidelurne = besoin >= 1
36
37          if ecqVidelurne:
38              besoin = 1
39
40          else:
41
42          for nomLtransf in lesLtransf:
43              voix = besoin * lesBull[nomLtransf]
44              lesDetails[lesLtransf[nomLtransf][0]][nomLtransf] += voix
45              lesVoix[lesLtransf[nomLtransf][0]] += voix
46
47              lesBull[nomLtransf] -= voix
48
49          if ecqVidelurne:
50              break
51          else:
52
53              lesElus.append(premier)
54              lesCand.remove(premier)
55              del lesDetails[premier]
56              del lesVoix[premier]
57              for nomLtransf in lesLtransf:
58                  lesLtransf[nomLtransf].remove(premier)
59          if len(lesCand) == 0:
60
61              break
62
63      if len(lesCand) <= 1:
64          break
65
66      voix, perdant = min([(voix, perdant) for
67                      perdant, voix in lesVoix.items()])
68
69      lesBull = lesDetails[perdant]
70        lesCand.remove(perdant)
71        del lesDetails[perdant]
72        del lesVoix[perdant]
73        for nomLtransf in lesLtransf:
74            lesLtransf[nomLtransf].remove(perdant)
75    if len(lesCand) == 1:
76        ultime = list(lesCand)[0]
77
78        for nomLtransf in lesLtransf:
79            voix = lesBull[nomLtransf]
80            lesDetails[ultime][nomLtransf] += voix
81            lesVoix[ultime] += voix
82        if lesVoix[ultime] > quorum / 2:
83            lesElus.append(ultime)
84        else:
```

```
85          lesCand.remove(ultime)
86          del lesVoix[ultime]
87          del lesDetails[ultime]
88          for nomLtransf in lesLtransf:
89              lesLtransf[nomLtransf].remove(ultime)
90
91      return lesElus
```

8.4 Programme Principal

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  """
5  Programme Principal
6  """
7
8  from ValidationSimulation1 import *
9  from VUTA_Simulation2 import *
10 from VUTA_v90401 import VUTA
11 from vote_basique import *
12 from VUTA_Simulation_Copule import *
13 """
14 Paramètres du programme
15 """
16
17 nbre_idees = 6 #Taille du vecteur d'idees
18 nbre_candidats = 50
19 nom_listes = ['l' + str(k) for k in range(nbre_candidats)] #liste des
       noms des listes
20 noms_candidats = [str(k) for k in range(nbre_candidats)]
21
22 nbre_sieges = 25
23 nbre_electeurs = 10000
24
25 dic_candidats_idees = dico_candidats_idees(nbre_candidats,
       noms_candidats,nbre_idees)
26 Liste_des_Idees_des_electeurs = liste_idees_electeurs(nbre_electeurs,
       nbre_idees)
27
28 ##SEUIL est le seuil de décision d'une loi.
29 SEUIL = 10
30
31
32 """
33 On définit les vecteurs d'idées pour les electeurs et les candidats
34 """
35
36 #idees_electeurs = liste_idees_electeurs(nbre_electeurs)
37
38 #On simule ici nos listes de transfert avec et sans la notoriété
39 simulation = simul(Liste_des_Idees_des_electeurs,noms_candidats,
       nom_listes,dic_candidats_idees,nbre_electeurs)
40
41 LTransf = simulation[0]
42 depouillement = simulation[1]
43
44 LTransf_SansNoto = simulation[2]
```

```
45 depouillement_sansNoto = simulation[3]
46 print(depouillement,depouillement_sansNoto)
47
48 simulation_sans_liste = simul_basique(nbre_electeurs,nbre_candidats,
       Liste_des_Idees_des_electeurs,noms_candidats,nom_listes,
       dic_candidats_idees)
49
50 """
51 On regarde qui est elu
52 """
53
54 elus_sansNoto = VUTA(LTransf_SansNoto, depouillement_sansNoto,
       nbre_sieges)
55 elus_avecNoto = VUTA(LTransf, depouillement, nbre_sieges)
56 elus_sansListe = basique(simulation_sans_liste,nbre_sieges,nom_listes)
57
58 print(elus_sansNoto)
59 print(elus_avecNoto)
60 print(elus_sansListe)
61
62 """
63 On crée les differents vecteur d'idees
64 """
65
66 #dic_candidats_idees = dico_candidats_idees(nbre_candidats)
67 dic_sans_noto = {k:dic_candidats_idees[k][0] for k in noms_candidats}
68
69 idees_elus_sansNoto = [dic_sans_noto[k] for k in elus_sansNoto]
70 idees_elus_avecNoto = [dic_sans_noto[k] for k in elus_avecNoto]
71 idees_elus_sansListe = [dic_sans_noto[k] for k in elus_sansListe]
72 #print(idees_candidats)
73
74
75 """
76 On observe les choix pour les electeurs pour un nombre nbre_lois de
       lois
77 """
78 electeur_Oui = 0
79 sansNoto_Oui = 0
80 avecNoto_Oui = 0
81 sansListes_Oui = 0
82 nbre_lois = 100
83 for k in range(nbre_lois):
84     loi = simulvectcopul()
85     #Ou simulvectg()
86     electeur_Oui += choix(Liste_des_Idees_des_electeurs,loi,SEUIL)
87     sansNoto_Oui += choix(idees_elus_sansNoto,loi,SEUIL)
88     avecNoto_Oui += choix(idees_elus_avecNoto,loi,SEUIL)
89     sansListes_Oui += choix(idees_elus_sansListe,loi,SEUIL)
90 print('Le Pourcentage de Oui des electeurs est de {:f}'.format(
       electeur_Oui/(nbre_electeurs*nbre_lois)))
91 print('Le Pourcentage de Oui des candidats sans Notoriété est de {:f}'.
       format(sansNoto_Oui/(nbre_sieges*nbre_lois)))
92 print('Le Pourcentage de Oui des candidats avec Notoriété est de {:f}'.
       format(avecNoto_Oui/(nbre_sieges*nbre_lois)))
93 print('Le Pourcentage de Oui des candidats sans Listes de Transfert est
        de {:f}'.format(sansListes_Oui/(nbre_sieges*nbre_lois)))
```

8.5 Simulation des vecteurs d'idées et simulation des élections

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from random import random
from math import sqrt,log,exp,pi,cos
import time
from VUTA_Simulation_Copule import *


"""
Première Partie : Simulation des différents vecteurs d'idées

"""

def boxmuller():
    #méthode de box muller permettant la simulation de variables aléatoires
    #Ces v.a suivent une loi normale centrée réduite
    r = sqrt(-2*log(random()))
    t = 2*pi*random()
    x = r*cos(t)
    return x

def simuleps():
    u = random()
    if u<1/2:
        eps = 0;
    else:
        eps = 1

    return eps

def dblegauss(m1,m2,sigma):
    #Pour simuler Z -> N (mu, sigma^2), on simule X -> N (0, 1)
    # on effectue ensuite la transformation: Z = mu + sigmaX.
    eps = simuleps()
    x = boxmuller()
    #On obtient donc la double gaussienne de notre modèle
    return eps*m1+(1-eps)*m2+sigma*x

def simulvectg(m1,m2,sigma,nbre_idees): #m1,m2 et sigma les paramètres de la gaussienne
    #on simule un vecteur aléatoire gaussien
    # Ce vecteur représente les idées d'un candiats ou électeurs
    vg = [0]*nbre_idees; #Initialisation du vecteur d'idées
    for i in range(nbre_idees):
        vg[i]= dblegauss(m1,m2,sigma)
        #simulation indépendante de variables dble gaussien
    return vg


def dico_candidats_idees(nbre_candidats,candidats,nbre_idees):
    #fonction qui renvoie un dictionnaire
    #Le dictionnaire associe un str candidat(key) à son vecteur d'idées liste(value)
```

```
53      dic_candidats_idees=dict()
54      for i in range(nbre_candidats):
55          cand = candidats[i]
56          vg = simulvectcopul()
57          notoriete = exp(log(2)*random()) # simulation de la notoriété
        dans [1,7]
58          dic_candidats_idees[cand]=[vg,notoriete]
59      return dic_candidats_idees
60
61  def liste_idees_electeurs(nbre_electeurs,nbre_idees): #Simule les idées
        des nbre_electeurs électeurs
62      LT = list()
63      for i in range(nbre_electeurs):
64          vg = simulvectcopul()
65          LT.append(vg)
66      return LT
67
68  """
69
70  Deuxième Partie : Production des listes de transfert
71
72  """
73  def dic_candidat_indice(listes_candidats):
74      #dictionnaire qui à chaque candidat associe un numéro
75      d = dict()
76      for k in range(len(listes_candidats)):
77          d[listes_candidats[k]] = k
78      return d
79
80  def invertdic(dictionnaire):
81      #fonction permettant d'inverser les clés et valeurs d'un
        dictionnaire
82      return dict([(v,k) for k,v in dictionnaire.items()])
83
84
85
86  def norme2(L1,L2):
87      #fonction permettant de calculer la norme euclidienne de deux
        vecteurs
88      # Ici les vecteurs sont des listes
89      #assert len(L1)==len(L2)
90      norme=0
91
92      for i in range(len(L1)):
93          norme+=(L1[i]-L2[i])**2
94
95      return sqrt(norme)
96
97
98
99  def dic_preferences_candidats(Liste_de_candidats,dic_tete_liste,
        candidats):
100     #fonction retournant un dictionnaire permettant d'associer chaque
        candidat
101     #de la liste_de_candidats à sa liste de transfert
102     #en fonction de ses préférences
103     dic_pref_candidat = dict()
104     for i in candidats: #on parcourt la liste des candidats
```

```
105         Liste_des_distances=dict() #on crée un dictionnaore vide
106         for j in Liste_de_candidats.keys(): #pour chaque candidat:
107             if j!=i: #Pourra nous être utile peut être plus tard
108                 #On ne prend pour calculer la distance entre deux
    candidats
109                 #Que leurs deux vecteurs d'idées
110                 ideeC1 = Liste_de_candidats[i][0]
111                 ideeC2 = Liste_de_candidats[j][0]
112 
113                 #On calcule la distance entre deux candidats d'après
    leurs idées
114                 #Si le candidat j est populaire cela va le rapprocher
    des
115                 #Autres candidats
116                 #On pondere par 1.2 pour candidat/candidat
117                 #On pondere par 2 pour electeur/candidat
118                 Liste_des_distances[j] = norme2(ideeC1,ideeC2)-1.2*
    Liste_de_candidats[j][1]
119                 #On ne veut pas de normes negatives
120                 #si la notoriété est telle que cela rend la norme
    negative
121                 #on passe cette norme à 0
122                 #Norme negative on ne s'en préoccupe pas
123                 if Liste_des_distances[j]<0:
124                     Liste_des_distances[j] = 0
125         l = sorted(Liste_des_distances.items(), key=lambda t: t[1])
126         #on classe ces distances dans l'ordre croissant
127 
128         l =[i]+ [l[i][0] for i in range(len(l))]
129         #on concatène le candidat en tête de liste avec sa la liste des
     candidats
130         #Elle est ainsi classée selon sa préférence
131         dic_pref_candidat[dic_tete_liste[i]]=l
132     #On retourne
133     return dic_pref_candidat
134 
135 """
136 La fonction suivante est la meme que la precedente
137 Toutefois, on ne prend pas en compte la notoriete
138 Les calculs restent les memes
139 Pour les commentaires, voir la fonction precedente
140 """
141 def dic_preferences_candidat_sansNoto(Liste_de_candidats,dic_tete_liste
    ,candidats):
142 
143     dic_pref_candidat = dict()
144     for i in candidats:
145         Liste_des_distances=dict()
146 
147         for j in Liste_de_candidats.keys():
148             if j!=i:
149 
150                 Liste_des_distances[j] = norme2(Liste_de_candidats[i],
    Liste_de_candidats[j])
151                 #Ici la difference est que l'on ne retranche pas la
    notoriete
152         l = sorted(Liste_des_distances.items(), key=lambda t: t[1])
153 
```

```
154         l =[i]+ [l[i][0] for i in range(len(l))]
155         dic_pref_candidat[dic_tete_liste[i]]=l
156 
157     return dic_pref_candidat
158 
159 
160 
161 """
162 On calcule une bonne fois pour toute la distance d'un electeur avec
    tous les
163 candidats
164 """
165 
166 def distances_electeurs_candidats_sansNoto(candidats,
    dic_candidats_idees,liste_idees_electeurs):
167     nbre_electeurs = len(liste_idees_electeurs)
168     dic = {str(k):{} for k in range(nbre_electeurs) } #Les entrées sont
     les electeurs
169     #Sous la forme 'i' ou i varie entre 0 et nbre_electeurs-1
170     #les valeurs sont des dictionnaires avec pour entrée un candidat et
     comme valeur la distance à ce candidat
171     for k in range(nbre_electeurs):
172         for j in candidats:
173             dic[str(k)][j] = norme2(liste_idees_electeurs[k],
    dic_candidats_idees[j])
174     return dic
175 
176 def distances_electeurs_candidats_avecNoto(candidats,
    dic_candidats_idees,liste_idees_electeurs):
177     nbre_electeurs = len(liste_idees_electeurs)
178     dic = {str(k):{} for k in range(nbre_electeurs) }
179     for k in range(nbre_electeurs):
180         for j in candidats:
181             dic[str(k)][j] = norme2(liste_idees_electeurs[k],
    dic_candidats_idees[j][0])-dic_candidats_idees[j][1]
182     return dic
183 
184 #Un électeur est identifié par sa position qui n'est utile que pour le
    code.
185 #Un électeur, une fois qu'il aura calculé la distance à tous les
    candidats
186 #regarde quelle liste est la plus proche de lui et cela determinera son
     choix.
187 def vote_un_electeur_sansNoto(dic_distances_electeur_candidats,listes,
    pos_electeur,dic_candidats_idees,liste_idees_electeurs,
    listes_transfert,candidats):
188     choix = str() #Initialisation du candidat
189     mini = float('Inf')
190 
191 
192     for k in listes:
193         norm = 0
194         for j in range(1,3):
195             #print(k,j)
196             candidat = listes_transfert[k][j]
197             #print(dic_distances_electeur_candidats,candidat)
198             distance = dic_distances_electeur_candidats[pos_electeur][
    candidat]
```

```
199                 norm+=distance/j
200             if norm <mini:
201                 mini = norm
202                 choix = k
203         return choix
204 
205 def vote_un_electeur_avecNoto(dic_distances_electeur_candidats,listes,
    pos_electeur,dic_candidats_idees,liste_idees_electeurs,
    listes_transfert,candidats):
206         choix = str() #Initialisation du candidat
207         mini = float('Inf')
208 
209         for k in listes:
210             norm = 0
211             for j in range(1,3):
212                 #print(k,j)
213                 candidat = listes_transfert[k][j]
214                 #print(dic_distances_electeur_candidats,candidat)
215                 distance = dic_distances_electeur_candidats[pos_electeur][
    candidat]
216                 norm+=distance/j
217             if norm <mini:
218                 mini = norm
219                 choix = k
220         return choix
221 
222 #On regarde ensuite les choix de tous les electeurs.
223 def votes_rapide_sansNoto(listes,dic_candidats_idees,
    liste_idees_electeurs,listes_transfert,candidats):
224         dic_distances_electeurs_candidats =
    distances_electeurs_candidats_sansNoto(candidats,
    dic_candidats_idees,liste_idees_electeurs)
225         nbre_electeurs = len(liste_idees_electeurs)
226         votes = list()
227         for k in range(nbre_electeurs):
228             votes.append(vote_un_electeur_sansNoto(
    dic_distances_electeurs_candidats,listes,str(k),dic_candidats_idees
    ,liste_idees_electeurs,listes_transfert,candidats))
229         return votes
230 
231 def votes_rapide_avecNoto(listes,dic_candidats_idees,
    liste_idees_electeurs,listes_transfert,candidats):
232         dic_distances_electeurs_candidats =
    distances_electeurs_candidats_avecNoto(candidats,
    dic_candidats_idees,liste_idees_electeurs)
233         nbre_electeurs = len(liste_idees_electeurs)
234         votes = list()
235         for k in range(nbre_electeurs):
236             votes.append(vote_un_electeur_avecNoto(
    dic_distances_electeurs_candidats,listes,str(k),dic_candidats_idees
    ,liste_idees_electeurs,listes_transfert,candidats))
237         return votes
238 
239 
240 
241 """
242 Dans le cas où il n'y a pas de liste de transfert
243 """
```

```
244 #Un electeur vote pour le candidat qui lui est le plus proche
245 def choix_electeurs_sansListes(nbre_electeurs,liste_idees_electeurs,
    candidats,dic_candidats_idees):
246     dic_sans_noto = {k:dic_candidats_idees[k][0] for k in candidats}
247     mini = nbre_electeurs*norme2([10 for k in range(len(
    liste_idees_electeurs))],[-10 for k in range(len(
    liste_idees_electeurs))])
248     choix = str()
249     for k in candidats:
250         #print(k,dic_sans_noto[k],liste_idees_electeurs)
251         norm = norme2(liste_idees_electeurs,dic_sans_noto[k])
252         if norm <= mini:
253             mini = norm
254             choix = k
255
256     return choix
257
258
259
260 #######
261 """
262
263 Troisieme Partie : Mise en place des fonctions pour retourner les
    resultats
264 sous une forme utilisable
265
266 """
267
268
269 #On compte ensuite les voix associées
270 def dic_comptage(listes):
271     d = dict()
272     for k in listes: # on initialise à  0 les voix associées aux
    candidats
273         d[k] = 0
274     return d
275 #On ne sait pas mais si on appelle deux fois la fonction dic_comptage
    de suite,
276 #cela pose probleme
277 def dic_comptage_sansnoto(listes):
278     d = dict()
279     for k in listes: # on initialise à  0 les voix associées aux
    candidats
280         d[k] = 0
281     return d
282
283
284 """
285 Les deux fonctions suivantes nous servent pour donner la forme de
    dictionnaire
286 en adequation avec le programme VUTA
287 """
288
289
290 def defdic_tete_liste(liste_de_transfert,candidats):
291
292     d = dict()
293     for k in range(len(candidats)):
```

```
294         #print(candidats[k])
295         d[candidats[k]] = liste_de_transfert[k]
296     #dictionnaire associant un candidat à sa liste de transfert A
    associé à lA
297     return d
298 
299 
300 def comptage_list(choix_electeursteurs,dic_comptage):
301     for k in choix_electeursteurs:
302         #choix_electeursteurs est issu de la fonction vote
303         #fonction qui donne les bulletins dépouillés
304         dic_comptage[k]+=1
305         #return dic_comptage
306     return dic_comptage
307 
308 
309 """
310 Programme Principal
311 
312 """
313 
314 """
315 Parametres de notre probleme
316 """
317 
318 
319 def simul(Liste_des_Idees_des_electeurs,candidats,listes,
    dic_candidats_idees,nbre_electeurs):#Pour la présentation en accord
     avec prgrm VUTA
320     dic_tete_liste = defdic_tete_liste(listes,candidats)
321     print('dic_tete_listeDone')
322     print(dic_tete_liste)
323 
324     #Issu de la simulation
325     #La notoriete y est integrée
326 
327     dic_sans_noto = {k:dic_candidats_idees[k][0] for k in candidats}
328     #En ne retenant que le vecteur d'idees on pourra comparer la
    simulation...
329     #avec et sans tenir compte de la notoriete
330 
331 
332 
333     #Issu de la simulation, on construit les vecteurs d'idees des
    electeurs
334     #Il sera utilise pour les deux simulations (avec et sans notoriete)
335     print('LTDone')
336     """
337     On commence tout d'abord par la simulation prenant en compte l'idee
     de notoriete
338     """
339 
340     Liste_des_Listes_de_transfert = dic_preferences_candidats(
    dic_candidats_idees,dic_tete_liste,candidats)
341     #renvoie la liste des listes de transferts
342     print('liste_des_listes_de_transfertDone')
343     t1 = time.time()
344     votes = votes_rapide_avecNoto(listes,dic_candidats_idees,
```

```
    Liste_des_Idees_des_electeurs,Liste_des_Listes_de_transfert,
    candidats)
345     t2 = time.time()
346     #On regarde donc le vote de chaque electeurs
347     print('votesDone')
348     print('temps vote avec noto',t2-t1)
349     dico_comptage = dic_comptage(listes)
350     #on initialise un dictionnaire pour le comptage des bulletins
351     #il a la forme suivante :
352     #{'lA': 0, 'lB': 0, 'lC': 0, 'lD': 0}
353     #Il entre en parametre de la fonction qui realise le depouillement
354
355     depouillement = comptage_list(votes,dico_comptage)
356     #on fait le depouillement
357     #ce depouillement nous renvoie le resultat sous la forme d'un
    dictionnaire
358     #ex : {'lB': 2460, 'lC': 1670, 'lA': 1937, 'lD': 3933}
359     #print(Liste_des_Listes_de_transfert)
360     #print(depouillement)
361
362     """
363     On compare ensuite avec les resultats ne prenant pas en compte la
    notoriete
364     """
365
366
367     Liste_des_Listes_de_transfert_sansnoto =
    dic_preferences_candidat_sansNoto(dic_sans_noto,dic_tete_liste,
    candidats)
368     t3 = time.time()
369     votes_sansnoto = votes_rapide_sansNoto(listes,dic_sans_noto,
    Liste_des_Idees_des_electeurs,
    Liste_des_Listes_de_transfert_sansnoto,candidats)
370     t4 = time.time()
371     dico_comptage_sansnoto = dic_comptage_sansnoto(listes)
372     #print(dico_comptage_sansnoto)
373     #print('noto',Liste_des_Listes_de_transfert)
374     #print('avec',Liste_des_Listes_de_transfert_sansnoto)
375     depouillement_sansnoto = comptage_list(votes_sansnoto,
    dico_comptage_sansnoto)
376     #print(Liste_des_Listes_de_transfert_sansnoto)
377     print('sans',depouillement_sansnoto)
378     print('avec',depouillement)
379
380
381     print('temps vote sans noto',t4-t3)
382     return ([Liste_des_Listes_de_transfert,depouillement,
    Liste_des_Listes_de_transfert_sansnoto,depouillement_sansnoto])
383
384
385 """
386 Nous faisons ici la simulation de la repartition des bulletins de vote
    pour
387 un vote simple ou un electeur ne choisit qu'un candidat
388 """
389 def simul_basique(nbre_electeurs,nbre_candidats,
    Liste_des_Idees_des_electeurs,candidats,listes,dic_candidats_idees)
    :
```

```
390     lesBull = {k:0 for k in listes}
391 
392     for k in range(nbre_electeurs):
393         choix = choix_electeurs_sansListes(nbre_electeurs,
        Liste_des_Idees_des_electeurs[k],candidats,dic_candidats_idees)
394         #print(choix,lesBull)
395         lesBull['l'+str(choix)]+=1
396 
397     return lesBull
```

8.6 Simulation des idées en utilisant la méthode des copules

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  
4  from scipy.integrate import quad
5  from math import *
6  import numpy as np
7  from numpy.linalg import cholesky
8  """
9  Première Partie : Simulation des différents vecteurs avec copules
10 
11 """
12 
13 #Fonction permettant de renvoyer la densité de la double bosse
14 def make_gauss(N, sigma, mu1,mu2):
15     return (lambda x: N/(sigma * (2*np.pi)**.5) *(
16             np.e ** (-(x-mu1)**2/(2 * sigma**2))+np.e ** (-(x-mu2)**2/(2
       * sigma**2))))
17 
18 
19 def F(x): #calcule la fonction de répartition de la double bosse
20     return quad(make_gauss(N=0.5, sigma=2, mu1=5,mu2 = -5), -np.inf, x)
       [0]
21 
22 
23 def FctRepartitionLoiNormale(x): #calcule la fonction de répartition de
       la loi normale N(0,1)
24     return quad(make_gauss(N=0.5, sigma=1, mu1=0,mu2 = 0), -np.inf, x)
       [0]
25 
26 
27 def intermediaire(x,y): #On introduit cette fonction car on va chercher
       le 0 de cette fonction
28     #Le 0 de cette fonction correspond à la fonction F^{-1}
29     return F(x)-y
30 
31 def dicho(a,b,y,prec): #fonction dichotomie classique pour trouver le 0
       de la fonction intermediaire
32     while b-a>prec:
33         c = (a+b)/2
34         if intermediaire(a,y)*intermediaire(c,y) <= 0:
35             b = c
36         else:
37             a = c
38     return (a+b)/2
39 
40 
```

```
41 def simulvectcopul():
42     C=np.array
       ([[1,0.9,0.9,0,0,0],[0.9,1,0.8,0,0,0],[0.9,0.8,1,0,0,0],[0,0,0,1,0,0.7],[0,0,0,0,1
43     #on définit une matrice de covariance pour nos 6 idées et on corrè
       le en fonction des différents types d'idées
44     #C1 : Migratoire C2: Interieur C3:Exterieur C4:Economie C5:
       Environnement
45     #C6 : Education
46     #on utilise la méthode décrite partie 5.
47     L=cholesky(C)
48     VectGauss=L.dot(np.random.normal(size=(6,1)))
49     precision=0.001
50
51     X1=VectGauss[0,0]
52     X2=VectGauss[1,0]
53     X3=VectGauss[2,0]
54     X4=VectGauss[3,0]
55     X5=VectGauss[4,0]
56     X6=VectGauss[5,0]
57
58     X1b=FctRepartitionLoiNormale(X1)
59     X2b=FctRepartitionLoiNormale(X2)
60     X3b=FctRepartitionLoiNormale(X3)
61     X4b=FctRepartitionLoiNormale(X4)
62     X5b=FctRepartitionLoiNormale(X5)
63     X6b=FctRepartitionLoiNormale(X6)
64
65     Y1=dicho(-10,10,X1b,precision)
66     Y2=dicho(-10,10,X2b,precision)
67     Y3=dicho(-10,10,X3b,precision)
68     Y4=dicho(-10,10,X4b,precision)
69     Y5=dicho(-10,10,X5b,precision)
70     Y6=dicho(-10,10,X6b,precision)
71
72     vc=list((Y1,Y2,Y3,Y4,Y5,Y6)) #on renvoie notre vecteur d'idées corr
       élées
73
74     return vc
75 """
76 Possibilité de faire un scatter plot
77 """
```

8.7 Simulation d'un système de vote basique

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 """
5 Ce programme reproduit un systeme de vote basique où l'on prend les
      premiers
6 candidats comme élus
7 """
8 #lesBull correspond au dictionnaire issue du depouillement qui associe
      à
9 #chaque candidat son nombre de voix
10 def basique(lesBull,nSieges,nom_listes):
11     lesElus = list()
```

```
    name_lists = list(nom_listes) #On fait une copie ici
    #print(lesBull)
    #dic_voix_candidats = dict([(v, k) for k, v in lesBull.items()])
    #print(dic_voix_candidats,'taille = ',len(dic_voix_candidats))

    while True:
        valeurs = lesBull.values()
        maxi = max(valeurs)
        #print(maxi)
        for k in name_lists:
            if lesBull[k]==maxi:

                lesElus.append(k.replace("l",""))
                name_lists.remove(k)
                del(lesBull[k])
        if len(lesElus)>=nSieges:
            break
    return lesElus

#lesBull = {'lA': 7976,
# 'lB': 9919,
# 'lC': 10587,
# 'lD': 26725,
# 'lE': 6058,
# 'lF': 17314,
# 'lG': 16009,
# 'lH': 5412};
#print(basique(lesBull,2))


#Scatter Plot
```

8.8 Codes pour la validation des Simulations en soumettant une loi

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Ce programme a pour but de tester le VUTA dans son ensemble
On va soumettre à l'assemblée issue du vote une loi.
Les deputes vont repondre par oui ou non
On obtiendra le pourcentage de Oui
On va soumettre aux electeurs qui ont élu l'assemblée le même projet de
    loi
On comparera ensuite les % pour regarder si l'assemblée issue du VUTA
est bien représentative
"""

#La loi est soumise manuellement
# La loi est aussi un vecteur de R^nbre_idees
#Si le projet de loi (par exemple une loi sur l'ecologie) ne concerne
    pas un
#Domaine, on mettra 0 à la coordonnée de cette loi

#C1 : Migratoire C2: Interieur C3:Exterieur C4:Economie C5:
    Environnement
#C6 : Education
```

```
21
22 #Pour determiner si un depute ou un electeur est en faveur d'une loi
23 #On va calculer la distance a ce projet de loi
24 #Si cette distance est inférieur à un seuil le depute votera Oui
25 from math import sqrt
26
27
28
29 def norme2(L1,L2):  #Dans ce cas L1 est une loi et L2 un vecteur d'idé
       es
30     #fonction permettant de calculer la norme euclidienne de deux
       vecteurs
31     # Ici les vecteurs sont des listes
32     assert len(L1)==len(L2)
33     norme=0
34
35     for i in range(len(L1)):
36         norme+=(L1[i]-L2[i])**2
37
38     return sqrt(norme)
39
40 #La fonction choix prend la liste des vecteurs d'idées des électeurs ou
       des élus
41 #La loi est un vecteur de R^{nbre_idees}
42 def choix(liste_idees,loi,SEUIL):
43     nbre_oui = 0
44     nbre_non = 0
45
46     for k in liste_idees:
47         dist_loi_idees = norme2(loi,k)
48         #On calcul la distance entre les idées d'un élu/electeur
49         if dist_loi_idees<SEUIL: #On compare la distance au SEUIL pour
       prendre la décision
50             nbre_oui+=1
51         else:
52             nbre_non+=1
53     #On retourne le nombre de personne qui votent pour Oui
54     return(nbre_oui)
```

Références

[PLA18] Frédéric PLANCHET. *MODÈLES FINANCIERS EN ASSURANCE ET ANALYSES DYNAMIQUES*. PhD thesis, 2017-2018.

[Wina] Le principe de condorcet, https ://www.youtube.com/watch ?v=hi89r4lqacc.

[Winb] Systèmes de scrutins, http ://www.vie-publique.fr/decouverte-institutions/institutions/approfondissements/differents-modes-scrutin-leurs-effets.html.