Numéro SCEI : 19138

TIPE

Doit-on changer notre système de vote présidentiel au profit du jugement majoritaire ?

PLAN

I- Introduction

II- Principe de Condorcet

1) Paradoxe de Condorcet
2) Théorème d'impossibilité d'Arrow

III- Etude comparée du jugement majoritaire et du scrutin uninominal majoritaire à deux tours

1) Le jugement majoritaire
2) Critère d'indépendance
3) Respect du principe de Condorcet
4) Critères supplémentaires

IV- Conclusion

I- Introduction : Qu’est ce qu’un bon système de vote ?

- Votants à égalité

- Possible de mettre en place à grande échelle

- Scrutin qui permet d’élire le candidat le plus légitime s’il existe : celui-ci doit satisfaire une majorité et contrarier dans une moindre mesure

II- Principe de Condorcet

Académie française

Nicolas de Condorcet (1743-1794), mathématicien, philosophe, homme politique et éditeur français des Lumières

Principe de Condorcet : « Si un candidat est préféré à tout autre par une majorité, alors il doit être élu »

Choix d'un candidat insuffisant : établissement d'une liste de préférences

Méthode de Condorcet : comparaison des candidats deux à deux pour élire un vainqueur de Condorcet

1) <u>Paradoxe de Condorcet</u>

Candidats A, B et C :

4 % : A>B>C

37 % : A>C>B

33 % : B>A>C

26 % : C>B>A

	A	B	C
A		41% : A>B	74%: A>C
B	59%: B>A		37%: B>C
C	26%: C>A	63%: C>B	

A est préféré à C

C est préféré à B

B est préféré à A

→ intransitivité de la relation de préférence

Probabilité d'un vainqueur de Condorcet

```
#Détermine le vainqueur de Condorcet (s'il existe)
def vainqueur_Condorcet(n):
    suffrage=gen_suff_norm(n)
    for i in range(n):
        winner=i
        adversaires=[k for k in range(n)] #Duel de i avec les autres
        adversaires.remove(i)             #candidats excepté lui
        for j in adversaires:
            v=vainqueurC(suffrage,i,j,n)
            if v!=i:
               winner='PARADOXE'
               break
        if winner==i:              #On vérifie si i est vainqueur
           return winner           #de Condorcet ou non
    return winner
```

Algorithmiquement

n=3 : P≈7,5%

n=4 : P≈16%

n=5 : P≈24%

n=6 : P≈31%

2) Théorème d’impossibilité d’Arrow

Pour un système où on classe les candidats, il est impossible de respecter les critères suivants :

1) **Non dictature** : les préférences d'un individu seul ne doivent pas déterminer le choix collectif
2) **Universalité** : la fonction de choix sociale doit être définie dans tous les cas de figure
3) **Unanimité** : si un candidat est préféré par la totalité des votants, cette préférence doit être celle de la société également
4) **Indépendance des options non pertinentes** : l'introduction d'un candidat supplémentaire ne doit pas modifier l'ordre relatif existant entre les autres candidats dans chaque bulletin

III-Etude comparée du jugement majoritaire et du suffrage uninominal majoritaire à deux tours (SUMDT)

1) Le jugement majoritaire

A- Description

- Mode de scrutin inventé par deux chercheurs du CNRS, Michel Balinski et Rida Laraki

- Méthode de meilleure médiane : système de vote par valeurs (appréciations verbales) qui se distingue par la détermination du gagnant par la médiane plutôt que la moyenne

- Gagnant : candidat ayant obtenu la meilleure mention majoritaire

Bulletin de vote du « jugement majoritaire »

Pour présider la France,
ayant pris tous les éléments en compte,
je juge en conscience que ce candidat serait :

Excellent	*Très Bien*	*Bien*	*Assez Bien*	*Passable*	*Insuffisant*	*à Rejeter*

Michel Balinski, Rida Laraki. Jugement majoritaire vs. vote majoritaire. 2012. .hal-00760250

B- Modélisation informatique

7 candidats, 1000 votants

e ▲	Type	Size	Value
0	str	1	très bien
1	str	1	bien
2	str	1	assez bien
3	str	1	passable
4	str	1	assez mauvais
5	str	1	mauvais
6	str	1	à rejeter

```
#Génère un suffrage de n votants
def gen_suffrage(n=7,card=1000):
    suffrage=np.zeros((card,n))
    for i in range(card):
        liste=gen_liste(n)
        suffrage[i]=liste
    return suffrage
```

	0	1	2	3	4	5	6
0	2	5	3	0	4	1	6
1	2	2	5	6	6	1	1
2	0	1	3	6	5	3	4
3	5	3	6	2	4	2	2
4	1	6	0	1	2	6	5
5	0	2	5	5	5	6	6
6	1	0	6	0	1	6	5
7	3	1	6	3	2	2	6
8	2	3	4	2	4	6	1
9	5	5	1	2	0	4	4
10	6	1	3	4	0	2	0

2) Critère d'indépendance

A- Répartition uniforme

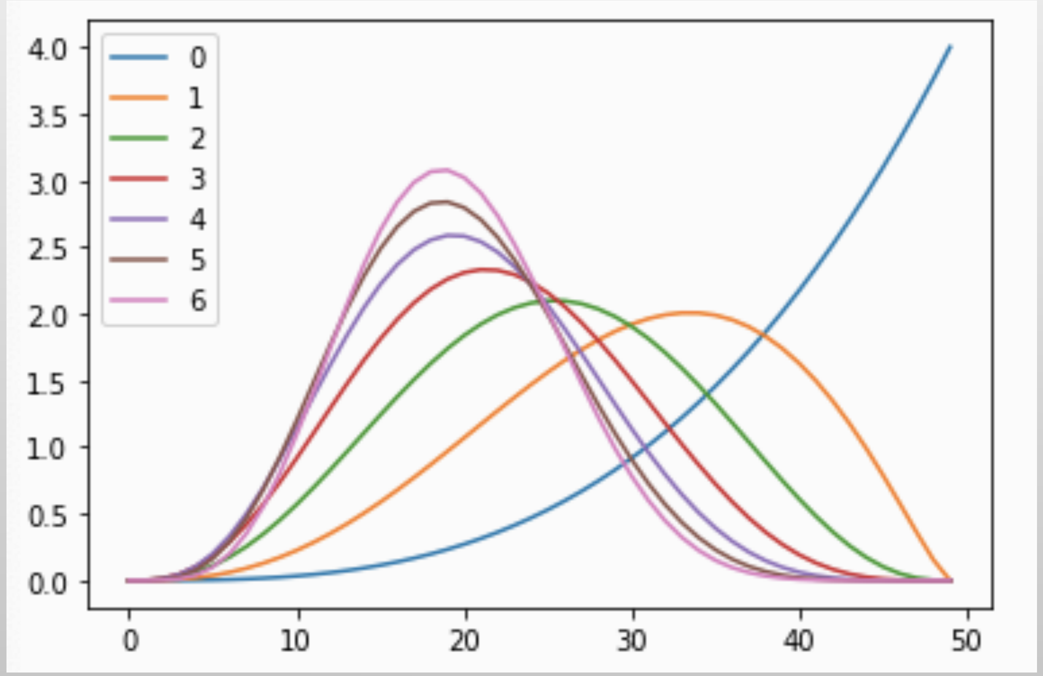
```
def gen_liste(n=7):
    L=np.zeros((1,n))
    for i in range(n):
        valeur=int(rd.uniform(0,7))
        L[0][i]=valeur
    return L
```

Retrait du candidat 0

win	tuple	2	([3, 'passable'], 1)
win_0	tuple	2	([3, 'passable'], 1)
win_1	tuple	2	([3, 'passable'], 3)
win_2	tuple	2	([3, 'passable'], 1)
win_3	tuple	2	([1, 'passable'], 1)
win_4	tuple	2	([3, 'passable'], 1)
win_5	tuple	2	([3, 'passable'], 1)
win_6	tuple	2	([3, 'passable'], 1)

B- Répartition bêta

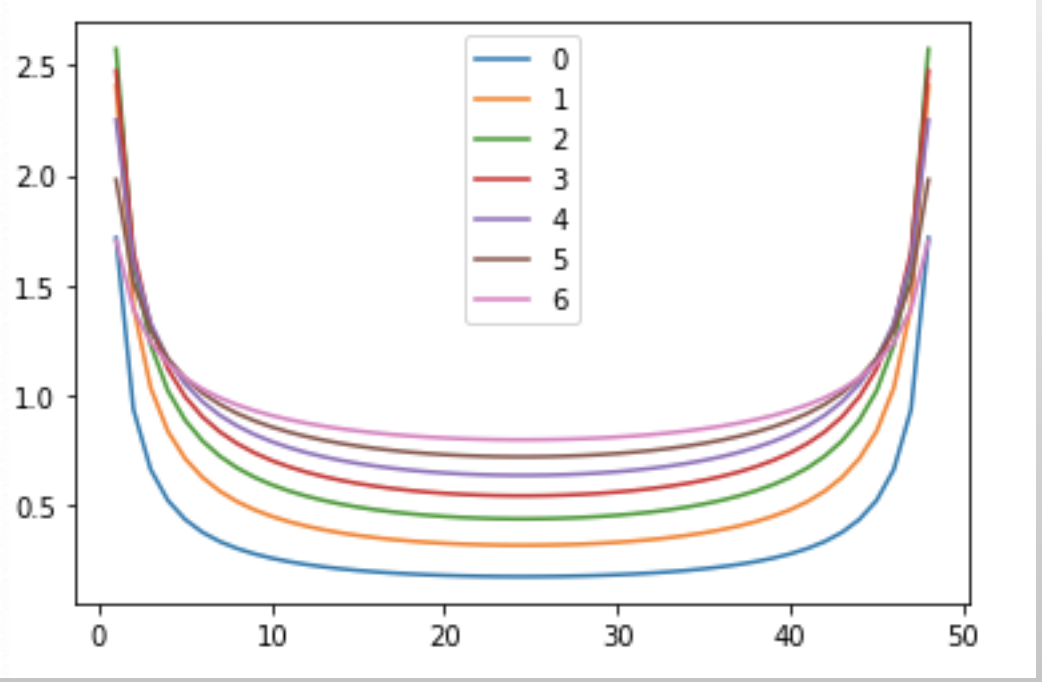
```
def gen_liste(n=7):
    L=np.zeros((1,n))
    for i in range(n):
        valeur=int(rd.betavariate(4-(0.3-0.1*i)*i,1+1.3*i)*6.99)
        L[0][i]=valeur
    return L
```



win	tuple	2	([5, 'assez bien'], 5)
win_0	tuple	2	([5, 'assez bien'], 6)
win_1	tuple	2	([5, 'assez bien'], 6)
win_2	tuple	2	([5, 'assez bien'], 5)
win_3	tuple	2	([5, 'assez bien'], 6)
win_4	tuple	2	([5, 'assez bien'], 6)
win_5	tuple	2	([6, 'assez bien'], 6)
win_6	tuple	2	([5, 'assez bien'], 5)

C- Répartition clivante

```
def gen_liste(n=7):
    L=np.zeros((1,n))
    for i in range(n):
        valeur=int(rd.betavariate(0.1*(i+1),0.1*(i+1))*6.99)
        L[0][i]=valeur
    return L
```



win	tuple	2	([0, 'assez bien'], 1)
win_0	tuple	2	([1, 'assez mauvais'], 2)
win_1	tuple	2	([0, 'assez bien'], 2)
win_2	tuple	2	([0, 'assez bien'], 1)
win_3	tuple	2	([0, 'assez bien'], 1)
win_4	tuple	2	([0, 'assez bien'], 1)
win_5	tuple	2	([0, 'assez bien'], 0)
win_6	tuple	2	([0, 'assez bien'], 1)

Exemple de dépendance des alternatives non pertinentes : élection présidentielle de 2002

Le jeu nuisible des candidatures fausse le résultat de l'élection

L'élection du 21 avril 2002.

Premier tour (16 candidats, 72% participation):

Chirac	Le Pen	Jospin	Bayrou	Laguiller	Chévènement
19,88%	16,86%	16,18%	6,84%	5,72%	5,33%

Mamère	Besancenot	Saint-Josse	Madelin	Hue	Mégret
5,25%	4,25%	4,23%	3,91%	3,37%	2,34%

(Pasqua)	Taubira	Lepage	Boutin	Gluckstein
–	2,32%	1,88%	1,19%	0,47%

Second tour (80% participation):

Chirac	Le Pen
82,21%	17,79%

Rida Laraki : présentation à l'Ecole Polytechnique

Bilan

- L'hypothèse de stabilité pour le jugement majoritaire se confirme : le retrait d'un candidat n'affecte pas l'issue du suffrage
- Pour le suffrage uninominal majoritaire à deux tours, le retrait d'un candidat peut changer l'identité du gagnant

3) Respect du principe de Condorcet

A- SUMDT

Candidats A, B et C
38 % : A>C>B
32 % : B>C>A
27 % : C>B>A
3 % : C>A>B

Premier tour : A avec 38% des voix
B avec 32% des voix

Deuxième tour : B avec 59% des voix

Avec la méthode de Condorcet :

	A	B	C
A		41% : A>B	38%: A>C
B	59%: B>A		32%: B>C
C	62%: C>A	68%: C>B	

Informatiquement :

Pourcentage de fois où le gagnant du SUMDT et le vainqueur de Condorcet (qu'il existe ou non) coïncident

count_beta	int	1	78
count_clivage	int	1	30
count_uni	int	1	42

paradoxe_beta	int	1	1
paradoxe_clivage	int	1	30
paradoxe_uni	int	1	28

B-Jugement majoritaire

Nombre de fois sur cent où le gagnant du jugement majoritaire et le vainqueur de Condorcet (qu'il existe ou non) coïncident

count_beta	int	1	14
count_clivage	int	1	30
count_uni	int	1	32

paradoxe_beta	int	1	2
paradoxe_clivage	int	1	23
paradoxe_uni	int	1	28

4) Critères supplémentaires

A- Critères restant du théorème d'Arrow

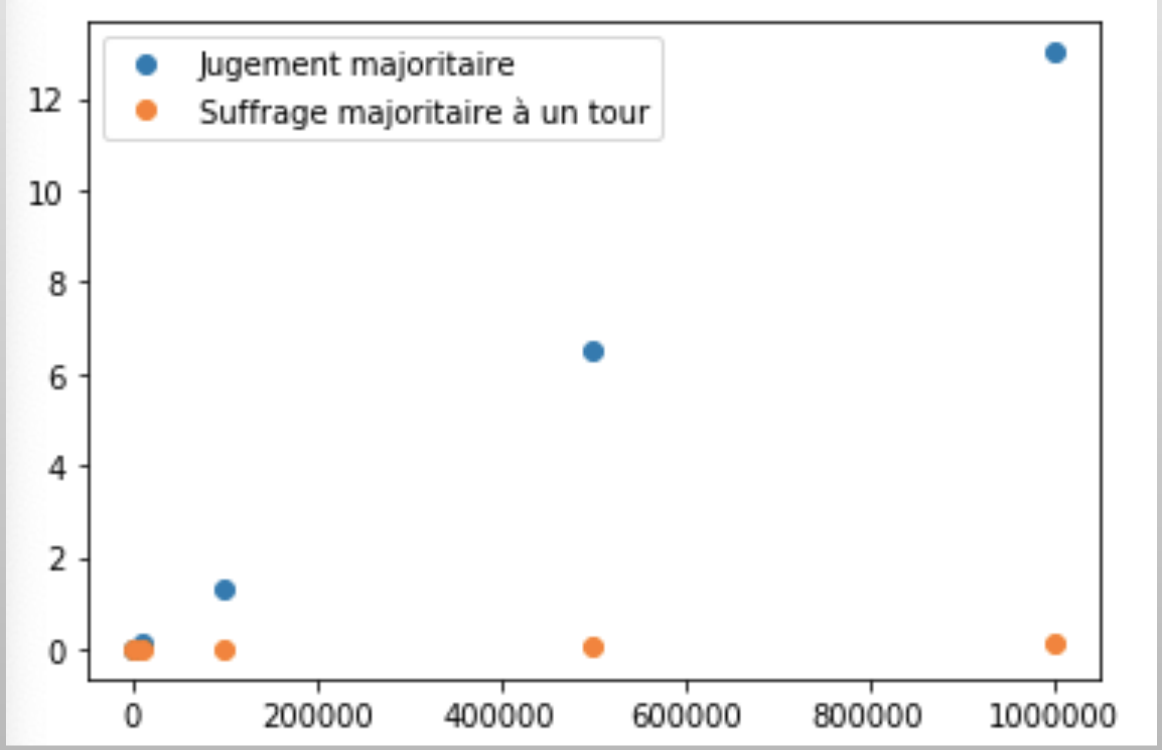
Pour les deux systèmes :

- **Critère de non dictature** ✓
- **Critère d'universalité** ✓
- **Critère d'unanimité** ✓

B- Temps d'exécution

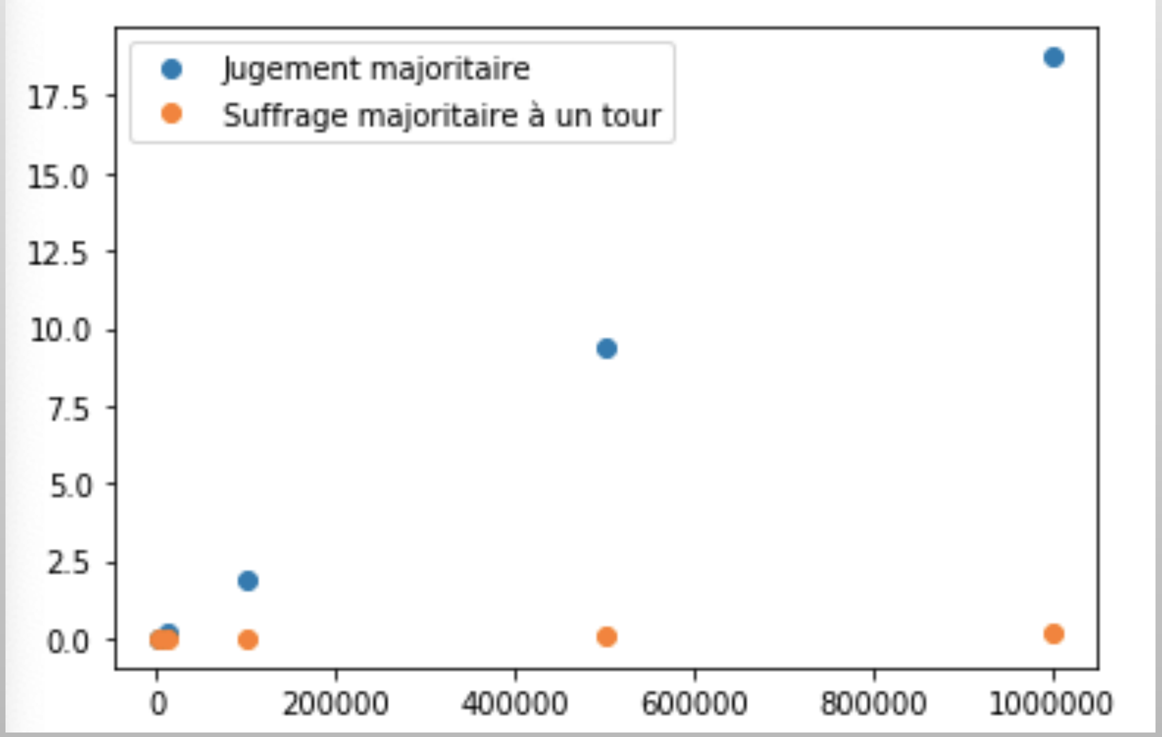
Détermination du vainqueur à partir d'un suffrage déjà établi

Nombre de candidats =7



a=1.3E-5

Nombre de candidats=10



a=1.825E-5

Le suffrage uninominal majoritaire à deux tours est plus facile à réaliser à grande échelle que le jugement majoritaire

Cependant, on constate une évolution linéaire pour le jugement majoritaire

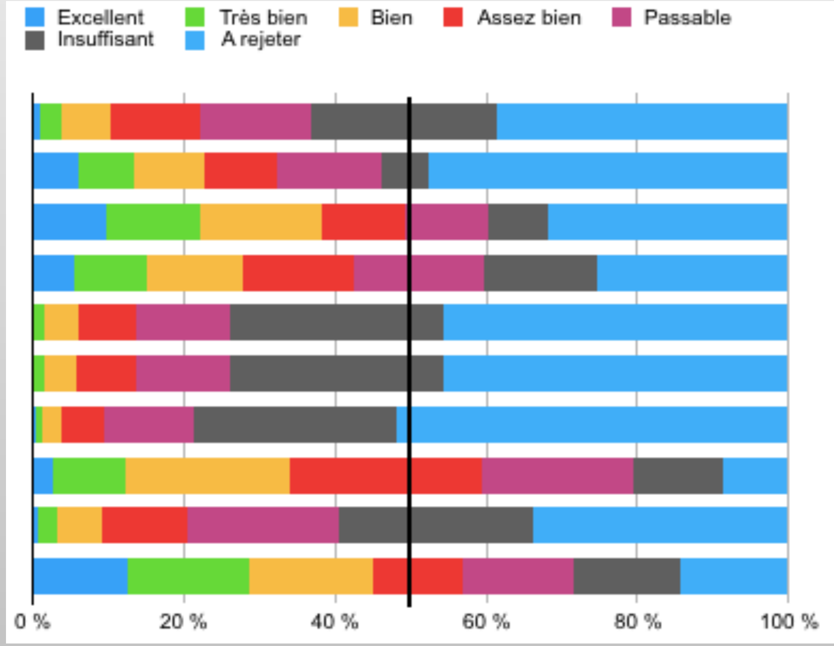
Donc pour 50 millions de votants on obtiendrait :

- 65s pour 7 candidats
- 91,25s pour 10 candidats

C- Le jugement majoritaire permet de quantifier la préférence

Exemple : élection de 2012

	Excellent	*Très Bien*	*Bien*	*Assez Bien*	*Passable*	*Insuffisant*	*à Rejeter*
Joly	0,81%	2,99%	6,51%	11,80%	14,65%	24,69%	38,53%
Le Pen	5,97%	7,33%	9,50%	9,36%	13,98%	6,24%	47,63%
Sarkozy	9,63%	12,35%	16,28%	10,99%	11,13%	7,87%	31,75%
Mélenchon	5,53%	9,50%	12,89%	14,65%	17,10%	15,06%	25,37%
Poutou	0,14%	1,36%	4,48%	7,73%	12,48%	28,09%	45,73%
Arthaud	0,00%	1,36%	4,48%	7,73%	12,48%	28,09%	45,73%
Cheminade	0,41%	0,81%	2,44%	5,83%	11,67%	26,87%	51,97%
Bayrou	2,58%	9,77%	21,71%	25,24%	20,08%	11,94%	8,69%
Dupont-Aignan	0,54%	2,58%	5,97%	11,26%	20,22%	25,51%	33,92%
Hollande	12,48%	16,15%	16,42%	11,67%	14,79%	14,25%	14,24%



Sondage commandé par Terra Nova et réalisé par OpinionWay

Classement Jugement Majoritaire	Au dessus Mention-Majoritaire *p*	La Mention-Majoritaire α±	En dessous Mention-Majoritaire *q*		Classement Scrutin Majoritaire	Les scores
1 Hollande	45,05%	*Assez Bien +*	43,28%		1 Hollande	28,7%
2 Bayrou	34,06%	*Assez Bien -*	40,71%		2 Sarkozy	27,3%
3 Sarkozy	49,25%	*Passable +*	39,62%		3 Le Pen	17,9%
4 Mélenchon	42,47%	*Passable +*	40,43%		4 Mélenchon	11,0%
5 Dupont-Aignan	40,57%	*Insuffisant +*	33,92%		5 Bayrou	9,1%
6 Joly	36,77%	*Insuffisant -*	38,53%		6 Joly	2,3%
7 Poutou	26,19%	*Insuffisant -*	45,73%		7 Dupont-Aignant	1,5%
8 Le Pen	46.13%	*Insuffisant -*	47,63%		8 Poutou	1,2%
9 Arthaud	24.83%	*Insuffisant -*	49,93%		9 Arthaud	0,7%
10 Cheminade	48,03%	*A Rejeter*	-		10 Cheminade	0,4%

D- Objection au jugement majoritaire : Non respect de la majorité

	50	50	1
A	Excellent	Assez mauvais	Assez mauvais
B	Assez bien	**A rejeter**	Passable

→ Assez mauvais

→ Passable

100 votants préfèrent A et 1 votant préfère B

Arrêter de considérer le plus grand groupe de satisfaits mais prendre en compte les plus satisfaits

« Tyrannie de la majorité »

Alexis de Tocqueville

Babelio

IV- Conclusion

Le jugement majoritaire :

✓

- N’est pas soumis au théorème d’impossibilité d’Arrow
- Donne une appréciation plus fine et plus détaillée des candidats
- Incite les électeurs à voter honnêtement et à penser leurs propres opinion
- N’encourage pas le vote utile

✗

- Non respect du principe de Condorcet
- Non respect de la majorité

Wikipedia

Kenneth Arrow
(1921-2017), économiste

« Les auteurs ont proposé une méthode de vote très intéressante pour **remédier aux défauts bien connus des méthodes standards**, comme le vote à la pluralité. Elle oblige les électeurs à exprimer leurs préférences d'une manière simple et facilement compréhensible, et les auteurs fournissent la preuve que le candidat choisi par leurs méthodes est une **sélection raisonnable**. Ce travail pourrait bien conduire à une **transformation utile de la pratique électorale**. »

Le jugement majoritaire est préférable au scrutin uninominal majoritaire à deux tours

ANNEXES

```python
import numpy as np
import random as rd

###MODELISATION : METHODE DE CONDORCET###


###GENERATION D'UN SUFFRAGE###

#Génère un classement aléatoire des candidats
def gen_listeC(n):
    L=[]
    while len(L)<n:
        cand=rd.randint(0,n-1)
        if cand not in L:
           L.append(cand)
    return L

#Pour n candidats, il y a n! classements possibles
def factorielle(n):
    if n==0:
        return 1
    else:
        return n*factorielle(n-1)

def gen_suffrageC(n):
    card=rd.randint(2,factorielle(n))  #Choix aléatoire du nb de
    suffrage=[]                        #classements distincts
    while len(suffrage)<card:
        liste=gen_listeC(n)            #Liste de listes regroupant
        if liste not in suffrage:      #tous les classements
           suffrage.append(liste)
    for i in suffrage:                #Ajout du nombre de votants
        i.append(rd.randint(1,100))   #pour ce classement
    return suffrage

#Normalisation pour que la somme des votants fasse 100
def gen_suff_norm(n):
    suffrage=gen_suffrageC(n)
    sum=0
    for i in suffrage:
        sum+=i[n]
    for i in suffrage:
        i[n]=i[n]*100/sum
    return suffrage
```

```python
###REALISATION D'UNE METHODE DE CONDORCET###

#Comptabilise les points des candidats dans un duel
def duelC(suffrage,i,j,n):
    voix_i=0
    voix_j=0
    for k in suffrage:              #Balayage de tous les
        if k.index(i)<k.index(j):   #classements
                voix_i+=k[n]        #Récupération du pourcentage
        else :                      #affecté à un classement
                voix_j+=k[n]
    return (voix_i,voix_j)


#Retourne le vainqueur d'un duel
def vainqueurC(suffrage,i,j,n):
    paire=duelC(suffrage,i,j,n)
    print(paire)
    if paire[0]>paire[1]:
        return i
    elif paire[0]<paire[1]:
        return j


#Détermine le vainqueur de Condorcet (s'il existe)
def vainqueur_Condorcet(n):
    suffrage=gen_suff_norm(n)
    for i in range(n):
        winner=i
        adversaires=[k for k in range(n)] #Duel de i avec les autres
        adversaires.remove(i)             #candidats excepté lui
        for j in adversaires:
            v=vainqueurC(suffrage,i,j,n)
            if v!=i:
               winner='PARADOXE'
               break
        if winner==i:              #On vérifie si i est vainqueur
           return winner           #de Condorcet ou non
    return winner

#Avec ce code, si un candidat a une égalité dans un duel, il ne peut
#pas être élu vainqueur
#Si le vainqueur de Condorcet existe, alors il est unique
```

```python
###MODELISATION :JUGEMENT MAJORITAIRE ET SUMDT###

#Différents modèles régissant la répartition des chiffre de 0 à 6
#attribués à chaque candidat
#Répartition alpha/beta : int(rd.betavariate(4-(0.3-0.1*i)*i,1+1.3*i)*6.99)
#int(rd.betavariate(4-0.3*i,3+0.3*i)*6.99)
#Répartition uniforme : int(rd.uniform(0,7))
#Clivage : int(rd.betavariate(0.1*(i+1),0.1*(i+1))*6.99)



#Dictionaire des mentions attribuables
D={'0':'très bien', '1':'bien', '2':'assez bien', '3':'passable',
   '4':'assez mauvais','5':'mauvais','6':'à rejeter'}



###GENERATION D'UN SUFFRAGE###

#Génère le suffrage d'un votant
def gen_liste(n):
    L=np.zeros((1,n))
    for i in range(n):
        valeur=int(rd.betavariate(0.1*(i+1),0.1*(i+1))*6.99)
        L[0][i]=valeur
    return L




#Génère un suffrage de n votants
def gen_suffrage(n=7,card=1000):
    suffrage=np.zeros((card,n))
    for i in range(card):
        liste=gen_liste(n)
        suffrage[i]=liste
    return suffrage
```

```python
###RECUPERATION DES RESULTATS###

#Compte pour chaque candidat le nombre de mentions reçues
def resultat(suffrage):
    _,n=np.shape(suffrage)
    resultats=np.zeros((n,7))
    for i in range(n):
        result_cand=list(suffrage[:,i])
        for j in range(7):
            resultats[i][j]=result_cand.count(j)
    return resultats


#Détermination de la mention majoritaire de chaque candidat
def mention(suffrage):
    card,n=np.shape(suffrage)
    result=resultat(suffrage)
    mediane=int(card/2)
    mentions=[]
    for j in range(n):
        count=0
        i=0
        while count<mediane:
            count+=result[j][i]
            i+=1
        mentions.append(i-1)
    return result,mentions
```

```python
###DETERMINATION DES GAGNANTS###

#Gagnant jugement majoritaire
def gagnant_JM(suffrage):
    _,n=np.shape(suffrage)
    result,mentions=mention(suffrage)
    mention_gagnante=min(mentions)
    mentions_litt=[]
    for i in mentions:
        x=str(i)              #Equivalent chiffre/mention
        mentions_litt.append(D[x])
    print(mentions_litt)
    favoris=[]
    for i in range(n):
        if mentions[i]==mention_gagnante:
            favoris.append(i)
    vainqueur=[favoris[0],D[str(mention_gagnante)]]
    m=len(favoris)
    if m==1:
        return vainqueur
    else:
        a_rejeter=list(result[:,6])                   #Départage des fav
        liste_favoris=[a_rejeter[i] for i in favoris] #Choix du
        mini=min(liste_favoris)                       #candidat avec le
        vainqueur[0]=a_rejeter.index(mini)            #moins de à rejeter
        return vainqueur
```

```python
#Gagnant suffrage uninominal majoritaire à deux tours
def gagnant_SUMDT(suffrage):
    suff_madt=[]
    n,m=np.shape(suffrage)
    for i in range(n):                   #On détermine à qui le votant
        liste=list(suffrage[i,:])        #a attribué la meilleure mention
        mini=min(liste)                  #parmi celles attribuées
        egalite=[]
        for k,l in enumerate(liste):
            if l==mini:                            #Liste des candidats ayant
                egalite.append(k)                  #cette mention et choix
        suff_madt.append(rd.choice(egalite)) #d'un gagnant parmi eux

    nb_voix=[]
    for i in range(m):                         #L'obtention d'une
     nb_voix.append(suff_madt.count(i))        #mention max donne
    print(nb_voix)                             #une voix au candidat

    #Détermination des deux vaignueurs du premier tour
    deuxieme_tour=[]
    maxi=max(nb_voix)
    deuxieme_tour.append(nb_voix.index(maxi))
    bis=[i for i in nb_voix if i!=maxi]
    maxi_2=max(bis)
    deuxieme_tour.append(nb_voix.index(maxi_2))
    print(deuxieme_tour)

    #Détermination du vainqueur du deuxième tour
    gagnant_1,gagnant_2=deuxieme_tour[0], deuxieme_tour[1]
    nb_voix=[0,0]                             #On garde le même suffrage
    for i in range(n):                        #Un candidat obtient la voix
          vote=list(suffrage[i,:])            #d'un votant s'il lui a
          if vote[gagnant_1]<vote[gagnant_2]: #donné une meilleure
              nb_voix[0]+=1                   #mention qu'à l'autre
          elif vote[gagnant_1]>vote[gagnant_2]:
              nb_voix[1]+=1
          else:
              i=rd.randint(0,1)
              nb_voix[i]+=1
    print(nb_voix)
    if nb_voix[0]>=nb_voix[1]:
        return gagnant_1
    else:
        return gagnant_2


def gagnants(suffrage):
    return suffrage,(gagnant_JM(suffrage),gagnant_SUMDT(suffrage))
```

```python
####EVALUATION DE LA STABILITE####

#Simulation du retrait d'un candidat de l'élection
def changement(suffrage,i):
    suffrage_bis=np.delete(suffrage,i,axis=1)
    _,win=gagnants(suffrage_bis)
    gagnant_JM,gagnant_SUMDT=win
    if gagnant_JM[0]>=i:                #La suppression d'une colonne
        gagnant_JM[0]+=1                #change les indices
    if gagnant_SUMDT>=i:
        gagnant_SUMDT+=1
    return (gagnant_JM,gagnant_SUMDT)


###RESPECT DU PRINCIPE DE CONDORCET###

#Comptabilise les points des candidats dans un duel
def duel(suffrage,i,j):
    voix_i=0
    voix_j=0
    m,n=np.shape(suffrage)
    for k in range(m):                     #Balayage de tous les
        if suffrage[k,i]<suffrage[k,j]:    #classements
                voix_i+=1
        elif suffrage[k,i]>suffrage[k,j]:
                voix_j+=1
        else:
            voix=rd.choice([voix_i,voix_j])
            voix+=1
    return (voix_i,voix_j)


#Détermination du vainqueur d'un duel
def vainqueur(suffrage,i,j):
    paire=duel(suffrage,i,j)
    if paire[0]>paire[1]:
        return i
    elif paire[0]<paire[1]:
        return j
```

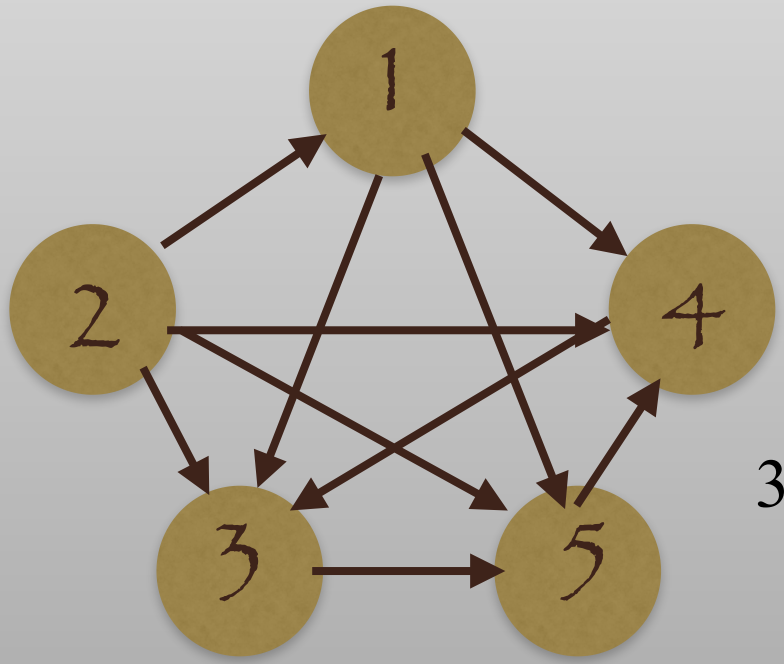
```python
def gagnant_C(suffrage):
    _,n=np.shape(suffrage)
    winner=0
    for i in range(n):
        winner=i
        bis=[k for k in range(n)] #Duel de i avec tous les autres
        bis.remove(i)             #candidats excepté lui
        for j in bis:
            v=vainqueur(suffrage,i,j)
            if v!=i:
               winner='PARADOXE'
               break
        if winner==i:                                  #On vérifie si i
           return (gagnant_JM(suffrage),winner)        #est vainqueur de
    return (gagnant_JM(suffrage),winner)               #Condorcet


def gagnant_C2(suffrage):
    _,n=np.shape(suffrage)
    winner=0
    for i in range(n):
        winner=i
        bis=[k for k in range(n)]
        bis.remove(i)
        for j in bis:
            v=vainqueur(suffrage,i,j)
            if v!=i:
               winner='PARADOXE'
               break
        if winner==i:
           return (gagnant_SUMDT(suffrage),winner)
    return (gagnant_SUMDT(suffrage),winner)

def count():
    count=0
    paradoxe=0
    for i in range(100):
        win=gagnant_C2(gen_suffrage())
        if win[0]==win[1]:
           count+=1
        if win[1]=='PARADOXE':
           paradoxe+=1
    return count, paradoxe
```

```python
###COMPARAISON DES TEMPS D'EXECUTION###

import time as tm
import matplotlib.pyplot as plt


#SUMDT non réalisé à partir d'un jugement majoritaire
def gen_suffrage_SU(n,card):
    suffrage=[]
    for i in range(card):
        suffrage.append(int(rd.betavariate(4-0.3*2,3+0.3*2)*n))
    return suffrage


def gagnant_SU(suffrage):
    n=max(suffrage)+1
    resultat=[]
    for i in range(n):
        points=suffrage.count(i)
        resultat.append(points)
    print(resultat)
    return suffrage,resultat.index(max(resultat))


def temps_JM(suffrage):
    t1=tm.clock()
    gagnant_JM(suffrage)
    t2=tm.clock()
    temps=t2-t1
    return temps
def temps_SU(suffrage):
    t1=tm.clock()
    gagnant_SU(suffrage)
    t2=tm.clock()
    temps=t2-t1
    return temps

n=10
Y=[1000,10000,100000,500000,1000000]
X1=[temps_JM(gen_suffrage(n,card)) for card in Y]
X2=[temps_SU(gen_suffrage_SU(n,card)) for card in Y]
plt.plot(Y,X1,'o',label='Jugement majoritaire')
plt.plot(Y,X2,'o',label='Suffrage majoritaire à un tour')
plt.legend()
```

Scrutin de Condorcet randomisé

- Election du vainqueur de Condorcet quand il existe sinon choix du gagnant à l’aide d’une loi de probabilité parmi un sous-ensemble de candidats

- Choix d’un sous-groupe de tête à l’aide des graphes orientés des duels



3/4/5 groupe de tête

- Association à chaque membre du sous- groupe d’une probabilité d’être élue
- Etablissement de loteries : L=(3:25%, 4:60%, 5:15%)
- Une loterie est préférée à une autre si le candidat tiré par par celle-ci est plus souvent mieux apprécié que le candidat tiré par la deuxième
- Détermination de la loterie de Condorcet

Théorème de Gibbard-Satterthwaite

Théorème de Gibbard (1973) : tout mécanisme de choix collectif est manipulable, dictatorial ou ne permet de choisir qu'entre deux options différentes

Gibbard-Satterthwaite : Pour un système de vote où on classe les candidats :

Lorsqu'une règle de vote est non manipulable, et qu'il existe trois ou plus de trois options à départager, alors cette règle est dictatoriale.

Méthode de départage initialement proposée par les créateurs du jugement majoritaire :

Méthode enlevant un vote par un vote

Départage des candidats ayant obtenu une même mention majoritaire :

- Retrait du vote de l'électeur ayant permis de déterminer cette mention
- Détermination de la nouvelle mention majoritaire des candidats
- Répétition de l'opération jusqu'à obtenir un départage
- Si un départage n'est pas possible, c'est que les candidats ont obtenu exactement les mêmes voix

Méthode Borda

Méthode formalisée par Jean-Charles de Borda en 1770

- Choix d'un nombre n inférieur ou égal au nombre de candidats
- Etablissement, pour chaque électeur, d'une liste de n candidats par ordre de préférence
- Le premier candidat de la liste reçoit n points, le deuxième (n-1) points etc

Vote par approbation

- Un électeur constitue une liste des candidats qu'il approuve
- La longueur de la liste est arbitraire
- Le candidat gagnant est celui ayant obtenu le plus de voix

Vote à second tour instantané / vote alternatif

- Classement des candidats par ordre de préférence
- On élimine le candidat ayant reçu le moins de voix et on redistribue ses voix à l'aide des classements
- Simulation de plusieurs tours jusqu'à obtention d'une majorité absolue pour un candidat

Variants du jugement majoritaire

Tout comme le jugement majoritaire, ce sont des méthodes de meilleure médiane
Ces scrutins s'en différencient par leur méthode de départage des inégalités :

- **Jugement usuel :** Mode de scrutin inventé en 2019 par le chercheur français en économie Adrien Fabre. Ordonne les candidats selon n= a+0.5*(p-q)/(1-p-q)   Avec : a=mention majoritaire
  p= part de votants ayant attribué une mention strictement supérieure que a
  q= part de votants ayant attribué une mention strictement supérieure que a

- **Jugement typique :** Ordonne les candidats selon n=a+p-q

- **Jugement central :** Ordonne les candidats selon n= a+0.5*(p-q)/(p+q)

Loi bêta

Famille de lois de probabilités continues et définies sur [0,1], paramétrées par deux variables α et β

Fonction densité de probabilité

$$f(x;\alpha,\beta) = \begin{cases} \dfrac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\,du} & \text{pour } x \in [0,1] \\ 0 & \text{sinon} \end{cases}$$

$$= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\, x^{\alpha-1}(1-x)^{\beta-1}\, \mathbb{1}_{[0,1]}(x)$$

$$= \frac{1}{\mathrm{B}(\alpha,\beta)}\, x^{\alpha-1}(1-x)^{\beta-1}\, \mathbb{1}_{[0,1]}(x)$$