# DM n°4 : Statistiques et Simulations — décembre 2017

Travailler par groupe de deux

## I] Âge des députés

Voici les répartitions des âges des députés, telles qu'elles apparaissent sur le site de l'Assemblée Nationale, pour les deux dernières législatures (les âges sont calculés juste après les élections et regroupés par classe).

| Députés | | [20;30[ | [30;40[ | [40;50[ | [50;60[ | [60;70[ | [70;80[ | [80;90[ | [90;100[ | total |
|---|---|---|---|---|---|---|---|---|---|---|
| 2017-2022 | effectifs | 24 | 97 | 165 | 186 | 91 | 13 | 0 | 0 | 576 |
| 2012-2017 | | 1 | 23 | 104 | 186 | 201 | 62 | 0 | 0 | 577 |

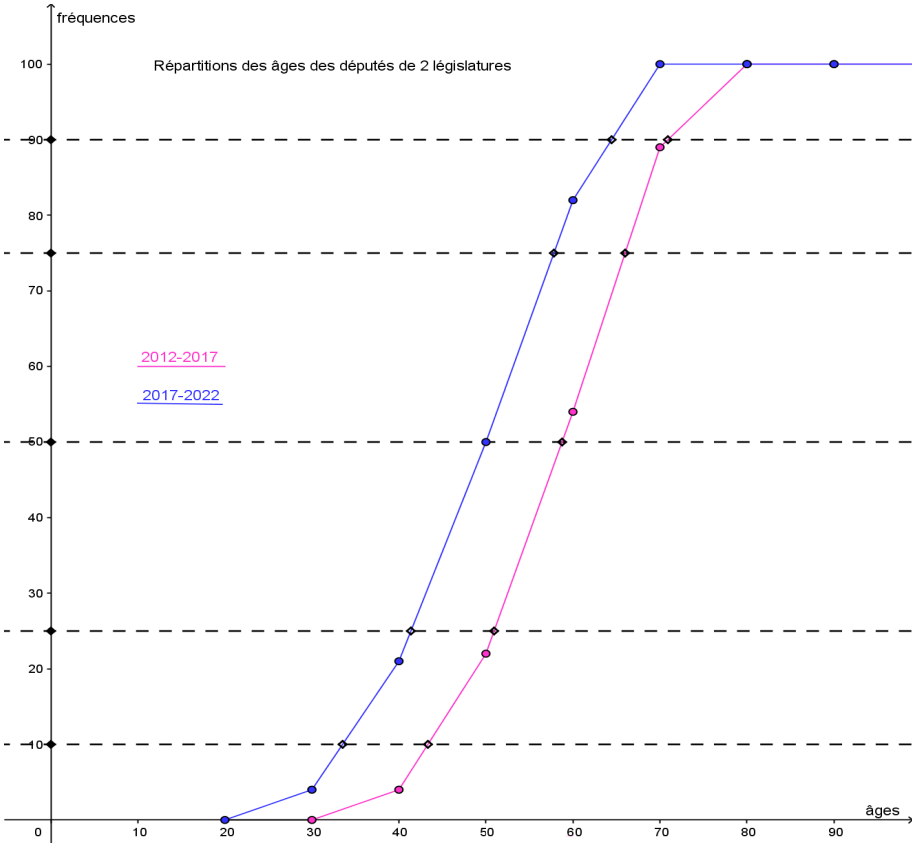
a) Représenter la courbe polygonale des fréquences cumulées pour les deux législatures (utiliser du papier millimétré et des couleurs) puis déterminer graphiquement la médiane $M$, les quartiles $Q_1$ et $Q_3$, les déciles $D_1$ et $D_9$ et les écarts inter-quartiles et inter-déciles relatifs. Dresser un tableau pour résumer ces informations. Commenter.

| Députés | [20;30[ | [30;40[ | [40;50[ | [50;60[ | [60;70[ | [70;80[ | [80;90[ | [90;100[ |
|---|---|---|---|---|---|---|---|---|
| 2017-2022 | 4% | 21% | 50% | 82% | 98% | 100% | 100% | 100% |
| 2012-2017 | 0% | 4% | 22% | 54% | 89% | 100% | 100% | 100% |

Voici le tableau des fréquences cumulées et la courbe polygonale représentant ces données :

Les paramètres statistiques ($D_1$, $Q_1$, M, $Q_3$ et $D_9$) que l'on peut lire sur ce graphique sont résumés dans le tableau ci-dessous, ainsi que les paramètres mesurant la dispersion relative des âges.

| Députés | D1 | Q1 | M | Q3 | D9 | (Q3-Q1)/M | (D9-D1)/M |
|---|---|---|---|---|---|---|---|
| 2017-2022 | 33,5 | 41,4 | 50,0 | 57,8 | 64,4 | 0,33 | 0,62 |
| 2012-2017 | 43,3 | 50,9 | 58,7 | 66,0 | 70,9 | 0,26 | 0,47 |
| différence | -9,8 | -9,56 | -8,67 | -8,19 | -6,47 | 0,07 | 0,15 |




La nouvelle législature est beaucoup plus jeune que la précédente : 8,7 années de différence sur les médianes ! Cet écart est considérable et il est plus marqué lorsqu'on considère les classes plus jeunes (9,8 années de différence sur les 1ers déciles) et moins marqué pour les classes plus âgées (6,5 années de différence sur les 9èmes déciles), le phénomène se retrouvant aussi sur les quartiles (une plus grande différence sur $Q_1$ que sur $Q_3$).

La dispersion des valeurs est à peu près stable, avec des écarts inter-quartiles voisins mais ils indiquent tout de même une augmentation de la dispersion des âges avec la dernière législature : l'écart inter-quartiles relatif passe de 0,26 à 0,33 et l'écart inter-déciles relatif passe de 0,47 à 0,62. L'écart inter-quartiles augmente en apparence deux fois moins que l'écart inter-déciles, mais c'est un effet de l'augmentation qui est calculée en valeurs absolues : 0,15 d'augmentation par rapport à 0,47 c'est une augmentation de 32% tandis que 0,07 d'augmentation par rapport à 0,26 c'est une augmentation de 27%. Les deux augmentations de la dispersion montrent donc un même phénomène. Cela montre un choix relativement bien marqué de rajeunir cette assemblée : plus de candidats jeunes sans doute et plus de voix pour ces candidats jeunes.

b) Déterminer, avec votre calculatrice, la moyenne et l'écart-type des âges des députés de ces deux dernières législatures. En 2012, le plus jeune député était une députée, Marion MARECHAL-LE PEN, née le 10 décembre 1989, tandis que le plus âgé était François SCELLIER, né le 7 mai 1936. En 2017, le plus jeune député est Ludovic PAJOT, né le 18 novembre 1993, tandis que le plus âgé est Bernard BROCHANT, né le 5 juin 1938. Déterminer la demi-étendue des âges des députés de ces législatures. Commenter.

Voici les calculs faits avec un tableur pour les âges des députés de la dernière législature.

| Députés | | [20;30[ | [30;40[ | [40;50[ | [50;60[ | [60;70[ | [70;80[ | [80;90[ | [90;100[ | total | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2017-2022 | ni=effectifs | 24 | 97 | 165 | 186 | 91 | 13 | 0 | 0 | 576 | | |
| | xi | 25 | 35 | 45 | 55 | 65 | 75 | 85 | 95 | | | |
| | nixi | 600 | 3395 | 7425 | 10230 | 5915 | 975 | 0 | 0 | 28540 | 49,55 | |
| | nixi² | 15000 | 118825 | 334125 | 562650 | 384475 | 73125 | 0 | 0 | 1488200 | 2583,68 | 11,34 |

On trouve une moyenne $\bar{x}=\frac{28540}{576}\approx 49{,}55$ ans et un écart-type $\sigma=\sqrt{\frac{1488200}{576}-49{,}55^2}\approx 11{,}34$ ans. L'écart-type relatif est donc de $\frac{\sigma}{\bar{x}}\approx\frac{11{,}34}{49{,}55}\approx 0{,}229$ . Pour ce qui est de l'étendue, le plus jeune député Ludovic PAJOT (né le 18/11/1993) avait 23 ans le jour de l'élection (le 18/06/17 pour le 2ème tour), le plus âgé Bernard BROCHANT (né le 5/06/1938) avait 79 ans le jour de l'élection. L'étendue est donc égale à $79-23=56$ ans pour les âges de cette assemblée et la demi-étendue vaut donc 28 ans.

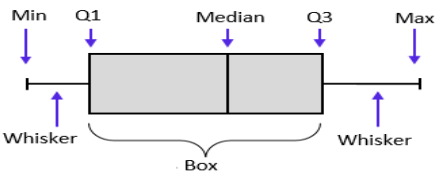
Voici le tableau pour les calculs des paramètres de la législature précédente. La moyenne est égale à 57,98 ans ( $\bar{x}=\frac{33455}{577}\approx 57{,}98$ ) et l'écart-type vaut 10,22 ans ( $\sigma=\sqrt{\frac{2000025}{577}-57{,}98^2}\approx 10{,}22$ ). L'écart-type relatif est donc de $\frac{\sigma}{\bar{x}}\approx\frac{10{,}22}{57{,}98}\approx 0{,}176$ .

| Députés | | [20;30[ | [30;40[ | [40;50[ | [50;60[ | [60;70[ | [70;80[ | [80;90[ | [90;100[ | total | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2012-2017 | ni=effectifs | 1 | 23 | 104 | 186 | 201 | 62 | 0 | 0 | 577 | | |
| | xi | 25 | 35 | 45 | 55 | 65 | 75 | 85 | 95 | | | |
| | nixi | 25 | 805 | 4680 | 10230 | 13065 | 4650 | 0 | 0 | 33455 | 57,98 | |
| | nixi² | 625 | 28175 | 210600 | 562650 | 849225 | 348750 | 0 | 0 | 2000025 | 3466,25 | 10,22 |

Le plus jeune député − la plus jeune députée de l'histoire de la République Française − était alors Marion Maréchal-LE PEN. Le 17 juin 2012, jour de l'élection, elle était âgée de 22 ans (née le 10/12/1989). Le plus âgé François SCELLIER (né le 7 mai 1936) avait, quant à lui, 76 ans ce jour. L'étendue est donc égale à $76-22=54$ ans pour les âges de la précédente assemblée et la demi-étendue vaut donc 27 ans.

Comparons les deux distributions des âges avec ces nouveaux paramètres. Les moyennes confirment le rajeunissement des députés, de 8,43 ans ($57{,}98-49{,}55=8{,}43$), que l'on avait constaté avec les médianes. Ce rajeunissement s'accompagne d'une plus grande hétérogénéité des âges puisque la dispersion augmente avec la nouvelle législature : de 17,6% de la moyenne, l'écart-type passe à 22,9% de la moyenne, ce qui représente une augmentation de plus de 30% ( $\frac{22{,}9-17{,}6}{17{,}6}\approx 0{,}3011$ ). L'étendue n'apporte pas grand chose au tableau : cet indicateur augmente tout de même de 2 ans. Cette augmentation va dans le même sens que l'augmentation des écart-types, mais est principalement due à l'âge relativement élevé du doyen des députés, Bernard BROCHANT. La très grande jeunesse de la précédente benjamine Marion Maréchal-LE PEN a fortement atténué l'augmentation de cette étendue qui, sans elle, aurait été de 10 ans (Marine BRENIER, la seconde benjamine de cette assemblée, née le 11 août 1986, était alors âgée de 30 ans).

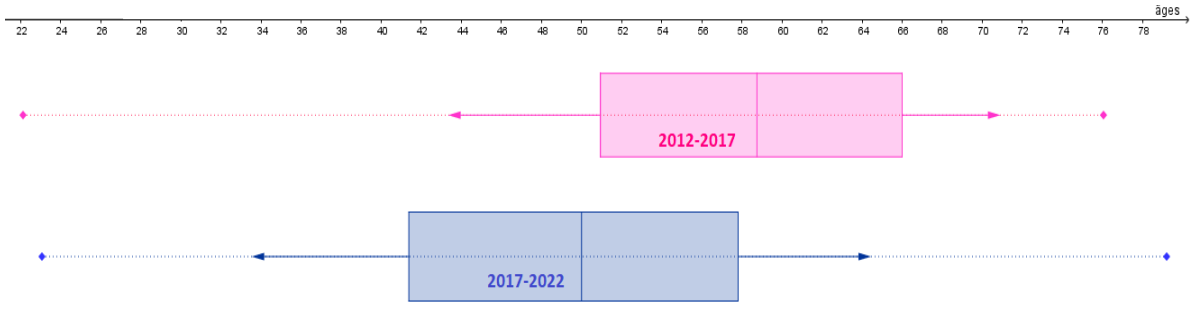
c) On parle de distribution dissymétrique lorsque la moyenne $M_y$ et la Médiane $M_e$ ne sont pas égales, et on précise généralement dans ce cas, si la distribution est étalée sur la droite ($M_y>M_e$) ou sur la gauche ($M_y<M_e$). La représentation par un diagramme de Tukey (*box plot* ou boîte à moustaches, voir notre illustration*) permet d'apprécier visuellement la dissymétrie, car dans le cas d'un étalement sur la droite, la médiane $M_e$ est plus proche de $Q_1$ que de $Q_3$. On peut aussi apprécier la dissymétrie avec le coefficient de Yule $C_Y=\frac{Q_1+Q_3-2M_e}{Q_3-Q_1}$ qui varie entre $-1$ et 1, étant positif dans le cas d'un étalement sur la droite.




Étudier avec ces outils, la dissymétrie des âges des députés. Commenter.

*Les extrémités des moustaches sont données par les déciles $D_1$ et $D_9$ quand on ne connaît pas les minimum et maximum.

Construisons les boîtes de Tukey des âges de ces deux assemblées. Nous avons indiqué les *extrema*, les âges du benjamin et du doyen (carrés aux extrémités de la ligne pointillée) ainsi que les déciles $D_1$ et $D_2$ (aux extrémités des flèches).



Visuellement, l'asymétrie n'est pas très forte. Il semble que les médianes soient du côté du 3ème quartile, signant un étalement vers la gauche des âges. On remarque aussi des flèches plus courtes à droite qu'à gauche, les écarts entre $Q_3$ et $D_9$ étant plus resserrés que ceux qu'il y a entre $Q_1$ et $D_1$. Cela va également dans le sens d'un étalement vers la gauche. Entre les deux législature, la différence d'asymétrie n'est pas flagrante. Ce qui apparaît bien davantage est le rajeunissement et l'augmentation de la dispersion.
Mesurons l'asymétrie avec les coefficients de Yule. Rappelons les résultats :

| | Q1 | M | Q3 | Yule |
|---|---|---|---|---|
| 2012-2017 | 50,9 | 58,7 | 66 | -0,03 |
| 2017-2022 | 41,4 | 50 | 57,8 | -0,05 |

$C_Y=\frac{Q_1+Q_3-2M_e}{Q_3-Q_1}=\frac{50,9+66,0-2\times58,7}{66,0-50,9}=-0,03$ dans le cas de la précédente législature et $C_Y=\frac{41,4+57,8-2\times50}{57,8-41,4}=-0,05$ pour la législature actuelle. Étant négatif dans les deux cas, cela confirme l'étalement sur la gauche, un peu plus prononcé dans le cas de la législature actuelle. Notons que ce coefficient mesure ce qui se voit sur la boîte à moustaches : le rapprochement de la médiane d'un des quartiles. Si $M=Q_3$, alors $C_Y=\frac{Q_1+Q_3-2Q_3}{Q_3-Q_1}=\frac{Q_1-Q_3}{Q_3-Q_1}=-1$ . Ici la médiane n'a pas fait tout le chemin vers $Q_3$. Pour $C_Y=-0,05=\frac{-1}{20}$, elle en a fait 5%. La différence $(Q_3-M_e)-(M_e-Q_1)\approx-\frac{Q_3-Q_1}{20}$ entre les deux différences vaut un vingtième, soit 5% de l'écart entre les deux quartiles. Le signe moins indique que c'est la différence avec $Q_1$ qui est la plus grande des deux.

## II] Marche aléatoire

a) Une puce se déplace sur un axe gradué, d'une unité à chaque fois, dans un sens ou dans l'autre, de façon aléatoire et équiprobable. On note $x$ l'abscisse de la puce après $n$ sauts, la puce étant partie de l'origine ($x$=0), et $d$ la distance entre l'origine et la position actuelle de la puce ($d$=|$x$|).
En s'inspirant de l'algorithme proposé à la partie C de l'exercice 37, page 266 de votre manuel, écrire un algorithme qui affiche le nombre $n$ de sauts effectués pour que la distance $d$ soit égale pour la 1ère fois à 1, 2, 3, etc. Transformer cet algorithme en programme pour la calculatrice[1] et donner un tableau donnant la suite de valeurs obtenue au bout de 1000 sauts pour dix marches aléatoires. Commenter les résultats de ce tableau.

Il suffit d'ajouter une instruction conditionnelle testant si $d$=|$x$| dépasse la valeur maximale.
Dans ce cas, on imprime la valeur de $i$ (le rang du saut) et on change la valeur du maximum en $d$.

```
#marche aléatoire sur un axe
#enregistrement de la distance maximale
from random import random
n=1000   #nombre de tirages
x=0      #abscisse de la puce
d=0      #distance maximum
for i in range(n) :
  a=random()
  if a<0.5 : x+=1
  else     : x-=1
  if abs(x)>d :
    d=abs(x)
    print("Distance {} atteinte au saut n° {}".format(d,i+1))
```

1 Les utilisateurs de la Numworks n'ont pas (encore) de générateur de nombre aléatoire. Ils utiliseront cet algorithme (voir au dos) :
$n$=xxxxxx [on initialise le nombre $n$ (la graine) par un nombre personnel à 6 chiffres, par exemple sa date de naissance]
$n$ = ($n$*1103515245+12345)/65536%32768 [on recalcule, à chaque passage dans la boucle, cet entier pseudo-aléatoire compris entre 0 et 32767 ; et on utilise $a$=$n$/32768 un nombre « réel » pseudo-aléatoire compris entre 0 (inclus) et 1(exclu)]

Voici donc un programme en Python qui traduit cet algorithme modifié :
Et voici les résultats de dix marches aléatoires. Certaines vont assez loin (76 pour la 5ème marche!), d'autres s'arrêtent relativement rapidement (22 pour la 4ème marche). En moyenne, au bout de 1000 sauts, la puce a été

| distance maxi | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Test 6 | Test 7 | Test 8 | Test 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 6 | 2 | 4 | 2 | 2 | 6 | 8 |
| 3 | 3 | 13 | 13 | 3 | 5 | 3 | 9 | 7 | 9 |
| 4 | 4 | 14 | 24 | 4 | 6 | 18 | 18 | 22 | 10 |
| 5 | 15 | 15 | 25 | 5 | 27 | 19 | 23 | 25 | 11 |
| 6 | 16 | 22 | 28 | 34 | 40 | 20 | 32 | 44 | 54 |
| 7 | 17 | 25 | 129 | 35 | 45 | 21 | 33 | 47 | 61 |
| 8 | 138 | 106 | 132 | 36 | 46 | 122 | 34 | 48 | 64 |
| 9 | 139 | 109 | 133 | 67 | 51 | 423 | 35 | 49 | 83 |
| 10 | 140 | 110 | 134 | 76 | 52 | 424 | 36 | 50 | 362 |
| 11 | 141 | 237 | 135 | 147 | 99 | 429 | 37 | 57 | 363 |
| 12 | 240 | 238 | 136 | 516 | 100 | 432 | 38 | 58 | 398 |
| 13 | 389 | 239 | 137 | 517 | 263 | 433 | 39 | 59 | 399 |
| 14 | 390 | 244 | 138 | 568 | 264 | 434 | 40 | 60 | 400 |
| 15 | 391 | 261 | 139 | 575 | 265 | 435 | 41 | 131 | 403 |
| 16 | 392 | 262 | 140 | 576 | 266 | 436 | 136 | 452 | 406 |
| 17 | 393 | 453 | 145 | 601 | 267 | 437 | 137 | 453 | 421 |
| 18 | 394 | 630 | 150 | 602 | 308 | 438 | 184 | 462 | 474 |
| 19 | 395 | 703 | 213 | 613 | 309 | 439 | 185 | 471 | 475 |
| 20 | 410 | 706 | 244 | 614 | 324 | 446 | 186 | 490 | 476 |
| 21 | 413 | 723 | 245 | 615 | 325 | 451 | 189 | 491 | 479 |
| 22 | 414 | 724 | 276 | 618 | 484 | 452 | 204 | 616 | 484 |
| 23 | 415 | 725 | 617 | | 497 | 657 | 205 | 617 | 499 |
| 24 | 494 | 726 | 712 | | 498 | 658 | 214 | 618 | 502 |
| 25 | 495 | 745 | | | 531 | 661 | 219 | 685 | 503 |
| 26 | 498 | 746 | | | 532 | 662 | 380 | 714 | 504 |
| 27 | 509 | 789 | | | 539 | 663 | 387 | 715 | 505 |
| 28 | 510 | 790 | | | 540 | 664 | 388 | 724 | 506 |
| 29 | 515 | 857 | | | 541 | 745 | 397 | 725 | 507 |
| 30 | 522 | 858 | | | 542 | 922 | 450 | 732 | 514 |
| 31 | | 859 | | | 543 | 923 | 451 | 733 | 669 |
| 32 | | 982 | | | 544 | 924 | 504 | 736 | 670 |
| 33 | | | | | 557 | 925 | 505 | 953 | |
| 34 | | | | | 558 | 964 | 506 | | |
| 35 | | | | | 567 | 971 | 509 | | |
| 36 | | | | | 636 | 972 | 532 | | |
| 37 | | | | | 645 | | 557 | | |
| 38 | | | | | 646 | | 562 | | |
| 39 | | | | | 653 | | 563 | | |
| 40 | | | | | 654 | | 564 | | |
| 41 | | | | | 665 | | 567 | | |
| 42 | | | | | 666 | | 568 | | |
| 43 | | | | | 667 | | 569 | | |
| 44 | | | | | 668 | | 574 | | |
| 45 | | | | | 669 | | 611 | | |
| 46 | | | | | 684 | | 612 | | |
| 47 | | | | | 685 | | 705 | | |
| 48 | | | | | 686 | | 706 | | |
| 49 | | | | | 687 | | 707 | | |
| 50 | | | | | 688 | | 708 | | |
| 51 | | | | | 701 | | 709 | | |
| 52 | | | | | 702 | | 710 | | |
| 53 | | | | | 703 | | 713 | | |
| 54 | | | | | 704 | | 716 | | |
| 55 | | | | | 705 | | 727 | | |
| 56 | | | | | 706 | | 728 | | |
| 57 | | | | | 707 | | 753 | | |
| 58 | | | | | 708 | | 756 | | |
| 59 | | | | | 721 | | 757 | | |
| 60 | | | | | 722 | | | | |
| 61 | | | | | 845 | | | | |
| 62 | | | | | 846 | | | | |
| 63 | | | | | 851 | | | | |
| 64 | | | | | 854 | | | | |
| 65 | | | | | 861 | | | | |
| 66 | | | | | 862 | | | | |
| 67 | | | | | 895 | | | | |
| 68 | | | | | 898 | | | | |
| 69 | | | | | 903 | | | | |
| 70 | | | | | 936 | | | | |
| 71 | | | | | 969 | | | | |
| 72 | | | | | 974 | | | | |
| 73 | | | | | 975 | | | | |
| 74 | | | | | 976 | | | | |
| 75 | | | | | 999 | | | | |
| 76 | | | | | 1000 | | | | |

jusqu'à une distance égale à (30+32+25+22+76+36+59+33+32+42)/10=38,7. Les progrès se font par saccades : parfois il y a un progrès à chaque saut (la puce a une série ascendante ou descendante) et parfois il faut attendre longtemps avant de progresser (la puce est revenue vers l'origine où elle alterne dans des abscisses basses).

b) Modifier l'algorithme pour que la puce continue à sauter tant qu'elle n'a pas atteint une distance $d_{max}$ fixée au départ. L'algorithme affiche toujours le nombre $n$ de sauts effectués pour que la distance $d$ soit égale pour la 1ère fois à 1, 2, 3, …, $d_{max}$. Exécuter le programme traduisant cet algorithme pour $d_{max}$=10 une dizaine de fois.
Lorsque $d_{max}$ double, le nombre $n$ de sauts effectués double t-il également?

Peut-on prévoir, à partir de ces résultats, le nombre moyen $n$ de sauts à effectuer pour $d_{max}$=20 ?
On remplace la boucle « pour » par une boucle « tant que ». Pour les résultats, on aurait pu se contenter de tronquer le précédent à 10 puisque l'on obtient ainsi le tableau désiré.

| distance maxi | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Test 6 | Test 7 | Test 8 | Test 9 | Test 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 6 | 2 | 4 | 2 | 2 | 6 | 8 | 2 |
| 3 | 3 | 13 | 13 | 3 | 5 | 3 | 9 | 7 | 9 | 3 |
| 4 | 4 | 14 | 24 | 4 | 6 | 18 | 18 | 22 | 10 | 6 |
| 5 | 15 | 15 | 25 | 5 | 27 | 19 | 23 | 25 | 11 | 7 |
| 6 | 16 | 22 | 28 | 34 | 40 | 20 | 32 | 44 | 54 | 10 |
| 7 | 17 | 25 | 129 | 35 | 45 | 21 | 33 | 47 | 61 | 67 |
| 8 | 138 | 106 | 132 | 36 | 46 | 122 | 34 | 48 | 64 | 68 |
| 9 | 139 | 109 | 133 | 67 | 51 | 423 | 35 | 49 | 83 | 143 |
| 10 | 140 | 110 | 134 | 76 | 52 | 424 | 36 | 50 | 362 | 144 |

Le programme Python qui exécute ce nouvel algorithme est donné cependant ci-dessous :

```
#marche aléatoire sur un axe - variante 1
#atteindre une certaine distance maximale
from random import random
n=0          #nombre de tirages
x=0          #abscisse de la puce
d=0          #distance maximum
dmax=10      #distance maximum à atteindre
while d<dmax:
  n+=1
  a=random()
  if a<0.5 : x+=1
  else     : x-=1
  if abs(x)>d :
    d=abs(x)
    print("Distance {} atteinte au saut n° {}".format(d,n))
```

Pour les dix marches, la moyenne du nombre de sauts nécessaire pour atteindre la distance $d$=10 est égale à :

(140+110+134+76+52+424+36+50+362+144)/10=152,8

Avec la même série, on peut calculer la moyenne du nombre de sauts nécessaire pour atteindre la distance $d$=20. Cette moyenne est égale à :

(410+706+244+614+324+446+186+490+476+620)/10=451,6

Visiblement, il ne suffit pas de doubler le nombre de sauts pour doubler la distance maximale atteinte. Ici, sur notre petit échantillon de 10 marches, il a fallu multiplier le nombre de sauts par 451,6÷152,8≈2,955, soit par 3 environ. Est-ce une bonne estimation ? Pour répondre à cette question, il faudrait effectuer cette étude sur plus de 10 marches. J'ai amélioré l'algorithme pour répondre à cette question. Voici mon nouveau programme et sa sortie pour 100 marches :

```
#marche aléatoire sur un axe - variante 2
#doubler la distance maximale à atteindre
from random import random
m=100
dmax=10    #distance maximum à atteindre
dmax2=dmax*2  #double de la distance maximum
sd,sd2=0,0
for i in range(m):
  x=0          #abscisse de la puce
  d=0          #distance maximum
  n=0          #nombre de tirages
  print("Marche numéro {} ".format(i+1))
  while d<dmax2:
    n+=1
    a=random()
    if a<0.5 : x+=1
    else     : x-=1
    if abs(x)>d :
      d=abs(x)
      if d%dmax==0:
        print("Distance {} atteinte au saut n° {}".format(d,n))
        if d==dmax : sd+=n
        else : sd2+=n
print("Nombre moyen de sauts pour atteindre une distance {} : {}".format(dmax,sd/m))
print("Nombre moyen de sauts pour atteindre une distance {} : {}".format(dmax2,sd2/m))
```

```
Marche numéro 1
Distance 10 atteinte au saut n° 92
Distance 20 atteinte au saut n° 446
Marche numéro 2
Distance 10 atteinte au saut n° 78
Distance 20 atteinte au saut n° 958
Marche numéro 3
Distance 10 atteinte au saut n° 78
Distance 20 atteinte au saut n° 224
  ....
Marche numéro 98
Distance 10 atteinte au saut n° 24
Distance 20 atteinte au saut n° 1798
Marche numéro 99
Distance 10 atteinte au saut n° 46
Distance 20 atteinte au saut n° 198
Marche numéro 100
Distance 10 atteinte au saut n° 24
Distance 20 atteinte au saut n° 56
Nombre moyen de sauts pour atteindre une distance 10 : 98.82
Nombre moyen de sauts pour atteindre une distance 20 : 404.7
```

Au regard de ces 100 marches, pour doubler la distance maximale il faut multiplier le nombre de sauts par

404,7÷98,82≈4,095 soit environ 4 fois. Un petit programme me donne les différentes valeurs moyennes ci-dessous pour 1000 marches et différentes valeurs de *dmax* obtenues en doublant à chaque fois. On y constate qu'il faut bien quadrupler le nombre de sauts pour doubler la distance maximale atteinte.

| *dmax* | 5 | 10 | 20 | 40 | 80 | 160 |
|---|---|---|---|---|---|---|
| Moyenne du nombre de sauts pour 1000 marches sur un axe | 24,86 | 97,06 | 393,17 | 1622,93 | 6255,59 | 25006,96 |
| Coefficient multiplicateur | - | 3,9 | 4,04 | 4,13 | 3,85 | 3,99 |

Tant qu'on y est, on peut chercher par combien il faut multiplier le nombre moyen de sauts pour tripler la distance. Je trouve alors qu'il faut multiplier le nombre de sauts par 866,06÷91,86≈9,43 soit environ 9 fois. Si j'essaie de réaliser *m*=1000 marches, j'obtiens un rapport 884,712÷98,596≈8,973.

```
#marche aléatoire sur un axe - variante 3
#tripler la distance maximale à atteindre
from random import random
m=100
dmax=10    #distance maximum à atteindre
dmax2=dmax*3  #triple de la distance maximum
sd,sd2=0,0
for i in range(m):
  x=0          #abscisse de la puce
  d=0          #distance maximum
  n=0          #nombre de tirages
  print("Marche numéro {} ".format(i+1))
  while d<dmax2:
    n+=1
    a=random()
    if a<0.5 : x+=1
    else     : x-=1
    if abs(x)>d :
      d=abs(x)
      if d%dmax==0:
        print("Distance {} atteinte au saut n° {}".format(d,n))
        if d==dmax : sd+=n
        if d==dmax2 : sd2+=n
print("Nombre moyen de sauts pour atteindre une distance {} : {}".format(dmax,sd/m))
print("Nombre moyen de sauts pour atteindre une distance {} : {}".format(dmax2,sd2/m))
print("Rapport de ces deux nombres moyens  : {}".format(sd2/sd))
```

```
...
Marche numéro 98
Distance 10 atteinte au saut n° 104
Distance 20 atteinte au saut n° 174
Distance 30 atteinte au saut n° 700
Marche numéro 99
Distance 10 atteinte au saut n° 20
Distance 20 atteinte au saut n° 368
Distance 30 atteinte au saut n° 424
Marche numéro 100
Distance 10 atteinte au saut n° 36
Distance 20 atteinte au saut n° 496
Distance 30 atteinte au saut n° 512
Nombre moyen de sauts pour atteindre une distance 10 : 91.86
Nombre moyen de sauts pour atteindre une distance 30 : 866.06
Rapport de ces deux nombres moyens  : 9.42804267363379
```

Quelle loi mathématique se cache derrière ces nombres ? Le nombre moyen de sauts pour atteindre une distance de 10 est environ de 100, pour atteindre une distance de 20 ce nombre est environ de 400 et pour atteindre une distance de 30 ce nombre est environ de 900. Il s'agit juste du carré de la distance. Cela explique que pour doubler une distance, il faut multiplier par 4=2² le nombre de saut et pour tripler il faut multiplier par 9=3². Pour décupler, il faut multiplier par 100=10². Pour atteindre la distance 100, en moyenne il faut 100²=10 000 sauts.

c) Au lieu de se déplacer sur un axe, la puce aléatoire se déplace dans un plan : partant de l'origine d'un repère orthonormé ($x$=0 ; $y$=0), elle peut se déplacer à chaque saut dans une des quatre directions (N, S, O ou E) de façon aléatoire et équiprobable. Cette fois, on note ($x$ ; $y$) les coordonnées de la puce après $n$ sauts. Écrire un algorithme permettant de simuler cette marche aléatoire dans le plan. La puce continue à sauter tant qu'elle n'a pas atteint (ou dépassé) une distance $d_{max}$ fixée au départ. On calculera la distance réelle depuis l'origine avec la formule $d=\sqrt{x^2+y^2}$ et on continuera à afficher le nombre $n$ de sauts effectués pour que la distance $d$ soit supérieure ou égale pour la 1ère fois à 1, 2, 3, …, $d_{max}$ (ici les valeurs de $d$ peuvent être irrationnelles). Programmer cet algorithme sur la calculatrice, puis exécuter le programme traduisant cet algorithme pour $d_{max}$=10 une dizaine de fois. Comparer les résultats avec la situation précédente.

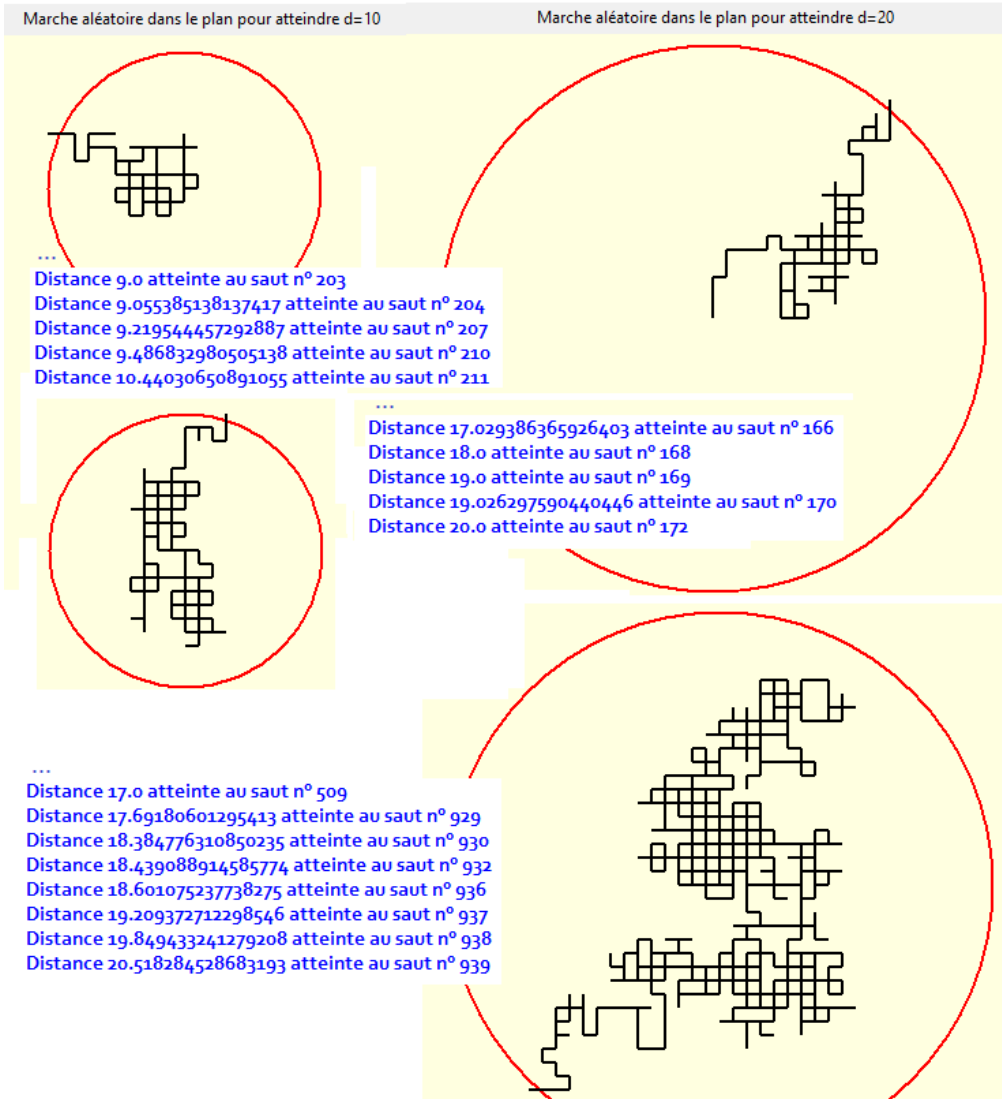
```
#marche aléatoire dans un plan
from math import sqrt
from random import random
n=0      #nombre de tirages
x=0          #abscisse de la puce
y=0          #ordonnée de la puce
d=0          #distance maximale atteinte
dmax=10    #distance à atteindre
while d<dmax :
  n+=1
  a=random()
  if a<0.25   : x+=1
  elif a<0.5  : x-=1
  elif a<0.75 : y+=1
  else        : y-=1
  if sqrt(x**2+y**2)>d :
    d=sqrt(x**2+y**2)
    print("Distance {} atteinte au saut n° {}".format(d,n))
```

```
Distance 1.0 atteinte au saut n° 1
Distance 1.4142135623730951 atteinte au saut n° 2
Distance 2.0 atteinte au saut n° 10
Distance 2.236067977499791 atteinte au saut n° 11
Distance 2.8284271247461903 atteinte au saut n° 12
Distance 3.605551275463989 atteinte au saut n° 13
Distance 4.242640687119285 atteinte au saut n° 14
Distance 5.099019513592785 atteinte au saut n° 22
Distance 6.082762530298219 atteinte au saut n° 47
Distance 7.071067811865475 atteinte au saut n° 48
Distance 8.062257748298551 atteinte au saut n° 49
Distance 9.055385138137417 atteinte au saut n° 50
Distance 10.049875621120889 atteinte au saut n° 53
```

Ici, il faut juste modifier le test du nombre aléatoire : selon les cas augmenter ou diminuer *x* ou *y*, les deux coordonnées de la puce. Le calcul de la distance est simple, mais on doit s'attendre à des valeurs différentes pour la progression des distances maximales, du fait des valeurs irrationnelles.
Voici les résultats de 5 marches aléatoires qui suivent ce principe. Sur ce petit échantillon, il faut en moyenne 14,8 étapes ((14×3+19+13)÷5=14,8) pour atteindre ou dépasser la distance de 10 unités. Cela s'obtient par une moyenne de 140,8 sauts ((116+343+75+119+51)÷5=140,8). Cela semble légèrement plus que ce qu'on peut obtenir lors d'une marche le long d'un axe (100 sauts pour atteindre la distance 10 en 10 étapes). Se déplacer

| étapes | distance maxi | Test 1 | distance maxi | Test 2 | distance maxi | Test 3 | distance maxi | Test 4 | distance maxi | Test 5 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1,00 | 1 | 1,00 | 1 | 1,00 | 1 | 1,00 | 1 | 1,00 | 1 |
| 2 | 2,00 | 2 | 1,41 | 4 | 1,41 | 6 | 2,00 | 6 | 1,41 | 6 |
| 3 | 2,24 | 5 | 2,00 | 8 | 2,24 | 7 | 2,24 | 7 | 2,00 | 7 |
| 4 | 2,83 | 6 | 3,00 | 9 | 2,83 | 8 | 3,16 | 8 | 2,24 | 8 |
| 5 | 3,61 | 7 | 4,00 | 10 | 3,61 | 11 | 3,61 | 15 | 2,83 | 15 |
| 6 | 4,47 | 8 | 5,00 | 11 | 4,47 | 12 | 4,47 | 16 | 3,61 | 16 |
| 7 | 5,00 | 11 | 6,00 | 12 | 5,00 | 15 | 5,00 | 19 | 4,24 | 19 |
| 8 | 5,66 | 12 | 6,08 | 13 | 5,39 | 51 | 5,10 | 20 | 5,10 | 20 |
| 9 | 6,40 | 15 | 7,07 | 14 | 6,08 | 55 | 6,00 | 26 | 6,08 | 26 |
| 10 | 7,07 | 16 | 7,28 | 23 | 7,07 | 66 | 6,08 | 27 | 7,07 | 27 |
| 11 | 7,81 | 17 | 7,62 | 24 | 8,06 | 67 | 7,07 | 28 | 8,06 | 28 |
| 12 | 8,60 | 106 | 8,54 | 25 | 9,06 | 70 | 7,21 | 42 | 9,05 | 42 |
| 13 | 9,00 | 115 | 9,49 | 26 | 9,49 | 74 | 7,28 | 51 | 10,05 | 51 |
| 14 | 10,00 | 116 | 10,05 | 343 | 10,44 | 75 | 7,62 | 112 | | |
| 15 | | | | | | | 8,54 | 113 | | |
| 16 | | | | | | | 8,60 | 116 | | |
| 17 | | | | | | | 9,22 | 117 | | |
| 18 | | | | | | | 9,90 | 118 | | |
| 19 | | | | | | | 10,63 | 119 | | |

dans un plan ralentit l'expansion car l'espace disponible dans un disque est plus grand que celui que contient le diamètre de ce disque. Il y a davantage de possibilités d'errance dans cet espace enclos par la frontière circulaire correspondant à une certaine distance maximale. Pour mieux visualiser ce point, voici une sortie graphique obtenue avec le module *tkinter* de Python.



Ce déplacement dans un plan semble donc un peu plus gourmand dans le temps. Pour s'écarter d'une distance *dmax* de l'origine, il faut consacrer en moyenne combien de sauts ? Un petit programme répond à cette question pour différentes valeurs de *dmax* :

| *dmax* | 5 | 10 | 20 | 40 | 80 | 160 |
|---|---|---|---|---|---|---|
| Moyenne du nombre de sauts pour 1000 marches dans le plan | 26,26 | 105,89 | 408,69 | 1642,31 | 6196,47 | 25926,85 |
| Coefficient multiplicateur | - | 4,03 | 3,86 | 4,02 | 3,77 | 4,18 |

On ne s'écarte donc vraiment pas de la loi mathématique précédente : il faut toujours un nombre de sauts environ égal au carré de la distance à atteindre.

# Génération d'un nombre aléatoire

*don't worry*, il n'y a rien à faire dans cette partie

Généralement, les langages informatiques disposent tous d'une fonction générant un nombre « réel » pseudo-aléatoire compris entre 0 (inclus) et 1(exclu). Cette fonction s'appelle souvent *random* (aléatoire en anglais) abrégé en *rand* parfois.

Voici, dans une syntaxe Python correcte (mettre à jour votre calculatrice Numworks pour la programmer) un générateur ultra simpliste de nombres pseudo aléatoires, noté GNPA. Il s'agit d'une fonction nommée *rand* qui prend en argument un nombre entier. Pour tester cette fonction *rand*, je vous propose cette petite fonction *alea* qui prend en argument un nombre entier de 6 chiffres appelé « graine » du GNPA (on peut essayer avec moins de chiffres mais le résultat n'est pas garanti). En entrant 123456 comme graine, c'est-à-dire en tapant *alea*(123456) dans la console Python, j'obtiens 498 qui montre que cette graine génère (au moins pour les 1000 premières valeurs) quasiment autant de nombres inférieurs à 0,5 que de nombres supérieurs à 0,5.

```
def rand(n) :
  return (n*1103515245+12345)/65536%32768

def alea(b) :
  n,p=b,0
  for i in range(1000) :
    n=rand(n)
    a=n/32768
    if (a<0.5) : p=p+1
  print(p)
```

On calcule ici l'image d'un nombre *n* par une fonction *rand* qui retourne un entier pseudo-aléatoire compris entre 0 et 32767.

Là, on teste la fonction *rand* en prenant une graine (*b*) et en calculant les 1000 valeurs suivantes de *n*. La valeur du nombre *a* qui est calculé à partir de *n* est testée à chaque fois pour comptabiliser les valeurs inférieures à 0,5 (normalement, si le GNPA fonctionne correctement, on doit trouver environ 500 valeurs).

Ce générateur peut être utilisé pour la question du DM. L'algorithme proposé à la partie C de l'exercice 37, page 266 de votre manuel, par exemple, se traduit par le programme Python suivant :

```
def rand(n) :
  return (n*1103515245+12345)/65536%32768


def marche(s) :
  n,x=123456,0
  for i in range(s) :
    n=rand(n)
    a=n/32768
    if (a<0.5) :
      x=x+1
    else :
      x=x−1
  print(x)
```

La fonction *marche* prend en argument le nombre entier *s* : le nombre de sauts que l'on veut faire faire à la puce (dans le livre cette variable est appelée *n* mais ici la lettre *n* est déjà utilisée par le générateur).
On initialise notre GNPA avec une graine (j'ai choisi 123456 mais on peut changer cette graine, sinon tout le monde obtiendra exactement la même chose) et la variable *x* est initialisée à 0 (*x* représente l'abscisse de la puce) .
L'instruction *n=rand*(*n*) sollicite le générateur à chaque tour de boucle tandis que *a=n*/32768 ramène le nombre pseudo-aléatoire trouvé entre 0 et 1.
Cet ensemble d'instructions est écrit dans un programme, nommé par exemple *aleatoire.py*. Lorsqu'on sort de l'éditeur, on trouve, à droite un petit onglet « ... » que l'on ouvre (en cliquant sur OK). On peut alors lancer l'exécution du script. La console s'ouvre et écrit *from aleatoire import* * ce qui signifie que tout c'est bien passé, le programme est prêt à être utilisé.
La fonction *alea*() se trouve en appuyant sur le bouton *var* ; écrire alors 1000 entre les parenthèses (on peut aussi écrire *alea*(1000) dans la console) et lancer l'exécution pour obtenir la position de la puce après 1000 sauts.

Bien sûr, tout cela vous paraîtra inutile et superflu lorsque vous disposerez, comme sur les calculatrices TI ou Casio, de toutes les fonctions du module *random* de Python (cela arrivera forcément, une calculatrice scientifique digne de ce nom ne pouvant se passer d'un bon GNPA). La fonction *randint* de ce module permettra alors de générer des entiers aléatoires compris entre deux valeurs : par exemple *randint*(0,3) génère de façon aléatoire des nombres de l'ensemble {0;1;2;3}. Faire ce que l'on a dit plus haut permet d'obtenir l'équivalent d'un nombre aléatoire de l'ensemble {0;1}. Si on veut 4 nombres entiers aléatoires équiprobables au lieu de 2 (ce qui est nécessaire dans la partie II-c du DM), il suffit d'imbriquer des tests portant sur le nombre *a* de l'intervalle [0;1[. Voici deux propositions, celle de droite utilisant l'instruction *elif* de Python qui est une contraction bien pratique de *else if* évitant les indentations successives :

```
if (a<0.5) :
  if (a<0.25) : [ce qui se passe quand a<0.25]
  else :        [ce qui se passe quand 0.25 <a<0.5]
else :
  if (a<0.75) : [ce qui se passe quand 0.5 <a<0.75]
  else :        [ce qui se passe quand 0.75 <a<1]
```

```
if (a<0.25) :   [ce qui se passe quand a<0.25]
elif (a<0.5) :  [ce qui se passe quand 0.25 <a<0.5]
elif (a<0.75) : [ce qui se passe quand 0.5 <a<0.75]
else :          [ce qui se passe quand 0.75 <a<1]
```