**Document d'accompagnement thématique du référentiel de formation**

**Inspection de l'Enseignement Agricole**

**Diplôme : Baccalauréat technologique**
« Sciences et technologies de l'agronomie et du vivant »
(STAV)

**Module : C4**
PRATIQUES MATHÉMATIQUES ET NUMÉRIQUES

**Thème :**
Exemples de situations pour la mise en œuvre du référentiel.

# Commentaires, recommandations pédagogiques,

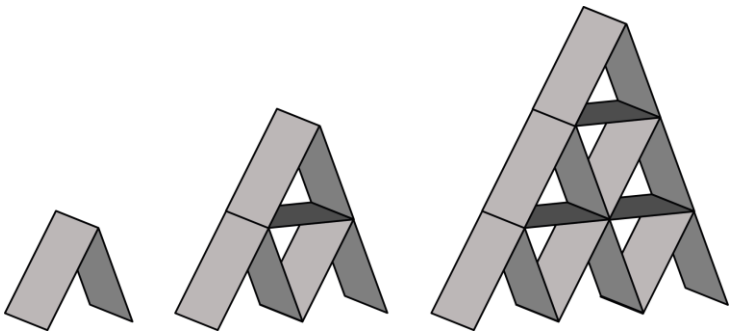
Ce document ne présente pas des activités directement exploitables mais des pistes de réflexion sur différentes thématiques du référentiel et qui doivent être adaptées en fonction des élèves. On peut les mener tout au long de l'année scolaire, voire du cycle en montrant le lien entre différents sous-objectifs. La recherche du sens des notions enseignées, notamment en filière technologique, est essentielle pour redonner du plaisir à la pratique des mathématiques. L'empilement de connaissances et de concepts n'ayant aucun lien entre eux est à éviter. Les exemples développés dans ce document ont pour but de montrer comment les notions peuvent se construire. Par ailleurs, l'un des moteurs du développement des mathématiques est la recherche de réponses à des problèmes issus de diverses situations concrètes. Plusieurs niveaux d'approches sont proposés afin de permettre au professeur d'aller plus ou moins loin, en fonction des contextes.

Conformément au référentiel, l'outil informatique est un large soutien à la compréhension des nouveaux concepts. Il y a plusieurs niveaux d'exigence et tous les exemples ne sont pas forcément à faire réaliser par les élèves, ils sont là aussi à destination du professeur pour lui permettre d'illustrer son enseignement.

## 2.1 Etudier des phénomènes discrets

### Un château de cartes

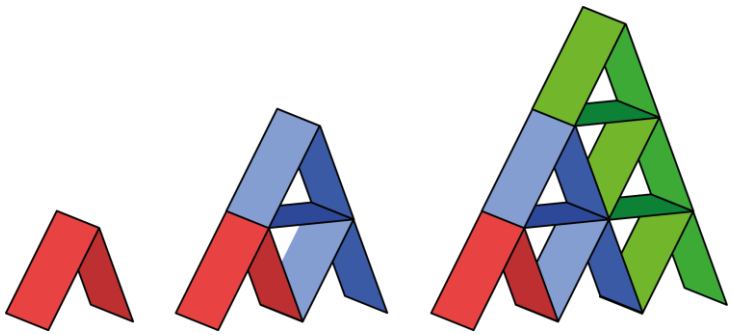
On considère la construction d'un château de cartes :



Soit la suite $(u_n)$ désignant le nombre de cartes utilisées dans la construction du château à l'étape $n$.

**Modélisation**

Le calcul des premiers termes de la suite $(u_n)$ permet d'obtenir la relation de récurrence $u_{n+1} = u_n + 3n + 2$. En effet, pour l'ajout d'un étage, il faut ajouter $n$ triangles complets donc $3n$ cartes plus le triangle incomplet de la base constitué de 2 cartes.



Grâce à un tableau de valeurs (réalisé à la main, avec un tableur, ...), on peut déjà déterminer le nombre d'étages réalisables avec deux jeux de 78 cartes (jeu de tarot).

**Programmation**

Le recours à l'algorithmique permet de donner le nombre d'étages pouvant être construits avec $p$ cartes, $p$ donné par l'utilisateur. Un exemple d'algorithme, avec sa traduction en langage Python :

$n \leftarrow 1$
$u \leftarrow 2$
Saisir $p$
Tant que $u \leq p$
    $u \leftarrow u + 3n + 2$
    $n \leftarrow n + 1$
Fin tant que
Afficher $n - 1$

```
n=1
u=2
p=int(input("donner le nombre de cartes disponibles"))
while u<=p:
    u=u+3*n+2
    n=n+1
print(n-1)
```

On peut alors utiliser le programme pour donner le nombre d'étages obtenu avec, par exemple, 100 paquets de jeu de tarot.

**Recherche de l'expression explicite de $u_n$ .**

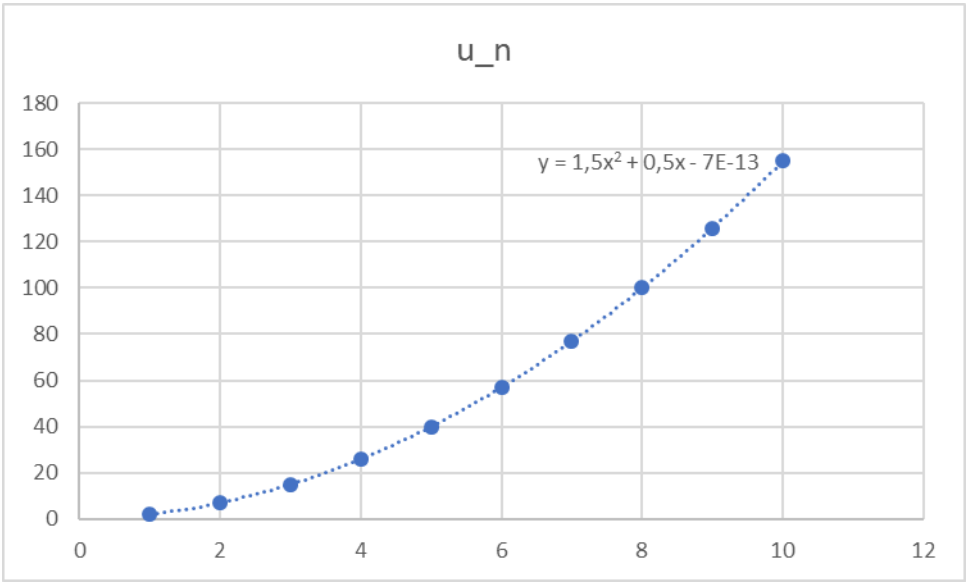
Nous avons déjà une formule récurrente pour la suite $(u_n)$ . Nous allons chercher une formule explicite en supposant qu'elle est de forme polynomiale. Cela revient à déterminer une fonction $f$ telle que $u_n = f(n)$ afin de calculer plus rapidement les termes de la suite $(u_n)$ .

Le tableur permet d'avoir les termes de la suite :

C3 | =B3+3*B2+2

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | |
| 2 | n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 3 | u_n | 2 | 7 | 15 | 26 | 40 | 57 | 77 | 100 | 126 | 155 |
| 4 | | | | | | | | | | | |

Le nuage de points tracé, on ajoute une courbe de tendance polynomiale de degré 2.




Une vérification a posteriori sur quelques valeurs permettra d'avoir un regard critique sur la valeur 7E-13 qui correspond à des erreurs d'approximation dans le traitement numérique.

Pour s'en convaincre, on peut utiliser un logiciel de calcul formel comme ***wxMaxima*** .

Fichier Édition View Cell Maxima Équations Algèbre Analyse Simplifier Tracé de courbes Numérique Aide

```
(%i1)   load("solve_rec");
(%o1)   C:\maxima-5.38.1\share\maxima\5.38.1_5_gdf93b7b_dirty\share\solve_rec\solve_rec
(%i8)   solve_rec(u[n+1]=u[n]+3*n+2, u[n],u[0]=0);
(%o8)   u_n = n(3n+1)/2
```

La résolution de cette question par le calcul, pour des élèves aguerris en calcul algébrique, peut s'opérer ainsi :

$u_n - u_{n-1} = 3(n-1)+2$

$u_{n-1} - u_{n-2} = 3(n-2)+2$

...

$u_1 - u_0 = 3\times(0)+2$

Par somme de ces égalités, on obtient : $u_n = 3\big((n-1)+(n-2)+...+0\big)+2n$ ce qui permet de réinvestir la somme des

$n$ premiers entiers pour obtenir $u_n = 3\dfrac{n(n-1)}{2} + 2n = \dfrac{3n(n+1)}{2}$ .

**Utilisation de l'expression littérale**

Il faut donc dans ce cas déterminer les valeurs de $n$ telles que $u_n \leq 156$ (cas de deux jeux de tarot, c'est une inéquation d'inconnue $n$).

Cela revient à résoudre l'inéquation $\dfrac{3}{2}n^2 + \dfrac{1}{2}n - 156 \leq 0$ .

$y = \dfrac{3}{2}x^2 + \dfrac{1}{2}x - 156$ est l'équation d'une parabole dirigée vers le haut car $a = \dfrac{3}{2} > 0$ .

À l'aide d'un solveur ou d'un logiciel de calcul formel, les valeurs approchées de l'équation $\dfrac{3}{2}x^2 + \dfrac{1}{2}x - 156 = 0$ sont –10,37 et 10,03.

Le plus grand entier naturel $n$ tel que $\dfrac{3}{2}n^2 + \dfrac{1}{2}n - 156 \leq 0$ est 10. Avec 156 cartes on peut donc construire un château comportant un maximum de 10 étages.

Par le même procédé la réponse pour 100 jeux de tarots se détermine en résolvant $\dfrac{3}{2}n^2 + \dfrac{1}{2}n - 7800 \leq 0$ .

On obtient $n \leq 71{,}9$ et on en déduit la construction complète de 71 étages.

## Le modèle de Verhulst.

Il est essentiel de prendre des situations concrètes qui permettent de s'initier à ce qu'est la modélisation et qui justifient l'étude d'outils mathématiques divers au service de la compréhension de phénomènes d'évolution. C'est le cas du modèle d'évolution de population de Verhulst.

Vers 1840, le mathématicien belge Verhulst a créé un modèle d'étude des populations. Comme tout modèle, la situation réelle est simplifiée en essayant d'imiter les conditions d'existence de la population.

Verhulst suppose que la population étudiée est limitée par une valeur maximale.

Il définit $u_n \in [0;1]$ comme le quotient de la population au temps $n$ par la population maximale possible.

Verhulst admet les deux principes suivants :

- $u_{n+1}$ (au temps $n+1$) est lié à $u_n$ (au temps $n$) . Le renouvellement dépend du nombre d'individus.
- $u_{n+1}$ est lié à $1-u_n$ . Le renouvellement dépend de la place qu'il reste.

Il considère donc la suite définie par $\begin{cases} k \in [0;4] \\ u_0 \in [0;1] \\ u_{n+1} = k \times u_n \times (1-u_n) \end{cases}$ pour tout $n$ entier non nul où $k$ est une constante qui dépend de la population étudiée. Pour cette situation, le modèle est admis et l'objectif est de l'exploiter.

**Illustration de ce modèle à l'aide d'un exemple plus contextualisé.**

Sur une île, le nombre d'arbres est limité par la surface de terre. Notons $P_{\max}$ le nombre maximal d'arbres possible sur l'île si c'était la seule espèce présente. Le cycle de reproduction de ces arbres étant annuel, le nombre d'arbres sur l'île au temps $n$, $n$ étant exprimé en années, est noté $P_n$. Soit $u_n = \dfrac{P_n}{P_{\max}}$, on a bien $u_n \in [0;1]$.

On admet que les deux principes de Verhulst sont acceptables pour notre contexte.

- $u_{n+1}$ (au temps $n+1$) est lié à $u_n$ (au temps $n$). Le renouvellement dépend du nombre d'arbres sur l'île. En effet, plus les arbres sont nombreux, plus ils ont de chance que leur pollen rencontre une fleur permettant de se reproduire.
- $u_{n+1}$ est lié à $1-u_n$. Le renouvellement dépend de la place qu'il reste. En effet, plus le nombre d'arbres est grand sur l'île, moins les jeunes pousses ont accès à la lumière d'où moins de nouveaux arbres peuvent se développer.

Par exemple, si une île peut héberger 200 arbres au maximum et qu'au bout de 5 ans (temps 5) il y en a 110.

On a $u_5 = \dfrac{110}{200} = 0,55$ , ce qui signifie que la population au temps 5 est à 55% de la population maximale atteignable.

**Appropriation de la situation**

Il importe que la situation soit appréhendée par des essais. On peut alors écrire une fonction en Python nommée «PnsurPmax» prenant comme paramètres les valeurs de $k$, $n$, $u_0$ et retournant la valeur de $u_n$ définie par

$\begin{cases} k \in [0;4] \\ u_0 \in [0;1] \\ u_{n+1} = k \times u_n \times (1-u_n) \end{cases}$ pour tout $n$ entier non nul où $k$ est une constante qui dépend de la population étudiée.

```
def PnsurPmax(k,n,u):
    for compteur in range(n):
        u = k*u*(1-u)
    return u
```

On obtient les valeurs suivantes pour $k = 2,7$, $n = 4$ et $u_0 = 0,25$.

| compteur | | | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|
| Rang de la suite | 0 | | 1 | 2 | 3 | 4 |
| u | 0,25 | | 0,506 | 0,675 | 0,592 | 0,652 |

**Compréhension des conditions**

Au-delà de la simulation, les mathématiques permettent de comprendre les conditions qu'impose le modèle, notamment pourquoi $k \in [0;4]$ .

La modélisation se fait donc par une suite exprimée sous la forme $u_{n+1} = k \times f(u_n)$ avec la fonction $f$ définie sur [0 ;1] par $f(x) = x(1-x)$. On établit, à l'aide de ce qui est connu sur les fonctions du second degré, le tableau de variation de $f$ qui est donnée sous forme factorisée donc on en déduit que les racines sont 0 et 1. La courbe représentative de $f$ est une parabole dont l'axe de symétrie est la droite d'équation $x = \dfrac{1}{2}$ (moyenne des deux racines). De plus $a = -1 < 0$ donc la parabole est dirigée vers le bas.

On peut alors en déduire que pour tout $x \in [0,1]$, on a $0 \leq f(x) \leq \dfrac{1}{4}$ .

| $x$ | $0$ | | $\frac{1}{2}$ | | $1$ |
|---|---|---|---|---|---|
| $f(x)$ | $0$ | ↗ | $\frac{1}{4}$ | ↘ | $0$ |

Ainsi, comme $u_n \in [0;1]$, on a $0 \leq f(u_n) \leq \dfrac{1}{4}$ d'où $0 \leq k\, f(u_n) \leq \dfrac{k}{4}$ ,

c'est-à-dire $0 \leq u_{n+1} \leq \dfrac{k}{4}$.

En prenant $0 \leq k \leq 4$ , on s'assure ainsi que si $u_n \in [0;1]$ alors $u_{n+1} \in [0;1]$ et en raisonnant de proche en proche on peut expliquer que pour tout entier naturel $n$ , $u_n \in [0;1]$ .

**Exploration numérique des propriétés du modèle**

L'étude générale des suites n'est pas un objectif du référentiel de STAV, mais la simulation permet de mettre en avant des propriétés de ce modèle.

La création d'une fonction « termes » donne la liste des $n$ premières valeurs de la suite pour certaines valeurs de $k$ et $u_0$.

```
def termes(k,n,u):
    valeurs = [u]
    for compteur in range(n):
        u = k*u*(1-u)
        valeurs.append(u)
    return valeurs
```

Si on garde les paramètres $k = 2{,}7$, $n = 4$ et $u_0 = 0{,}25$, il faut écrire « print(termes(2.7,4,0.25)) » et le programme affiche : [0.25, 0.5062500000000001, 0.67489453125, 0.5924121379348755, 0.651941991258225]

La création d'une fonction « termes » permet de tracer un nuage de points à partir de deux listes, une liste d'abscisses et une liste d'ordonnées. On importe le module « matplotlib.pyplot » pour réaliser les grahiques.

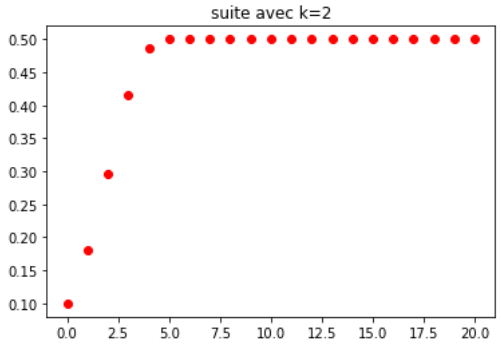
```
from matplotlib.pyplot import*

def trace(x,y,k):
    plot(x,y,"ro")
    title("suite avec k="+str(k))
    show()
```

**Évolution du modèle suivant les paramètres**

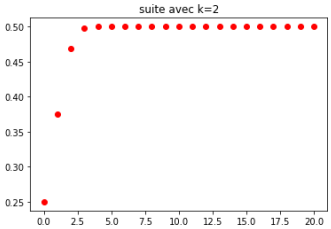
Il s'agit maintenant de faire de la simulation pour étudier le comportement de la suite de Verhulst en fonction de différentes valeurs de $k$ et de $u_0$. Il est important de ne pas oublier qu'on étudie l'évolution d'une population d'arbres.

En utilisant « trace(range(21),termes(2,20,0.1),2) », on affiche le graphique ci-contre :

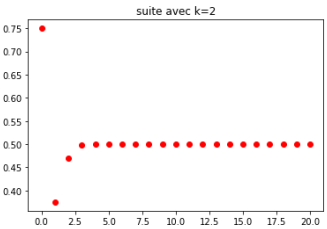
La suite semble croissante jusqu'à $n = 5$ puis constante, toujours égale à 0,5.



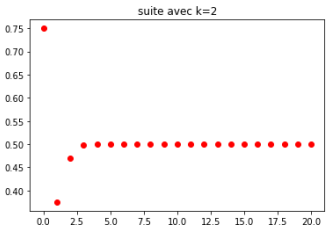

On peut changer les conditions initiales, avec le même cas, par exemple :




$u_0 = 0{,}25$




$u_0 = 0{,}75$



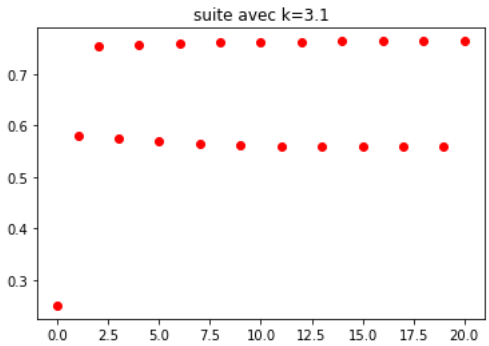

$u_0 = 0{,}99$

On peut remarquer que :

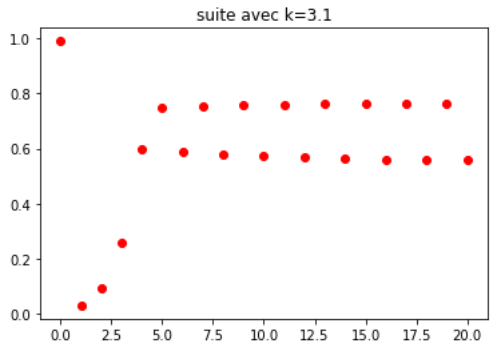
- si on choisit $u_0 > 0{,}5$ on obtient $u_1 < 0{,}5$.
- à partir de $n = 1$, les suites sont très similaires : croissantes puis rapidement constantes.
- elles tendent toutes vers 0,5, donc quelle que soit la population d'arbres au départ, elle finit par occuper la moitié de l'espace maximal.

En changeant la valeur de $k$ :

- **$k = 3{,}1$**



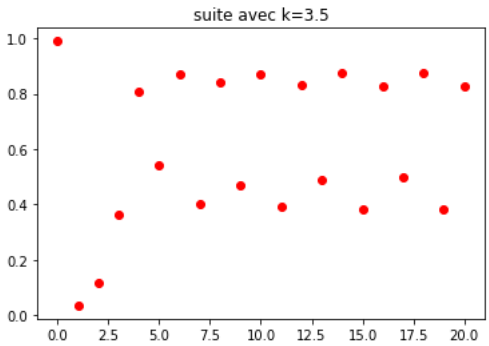

$u_0 = 0{,}25$

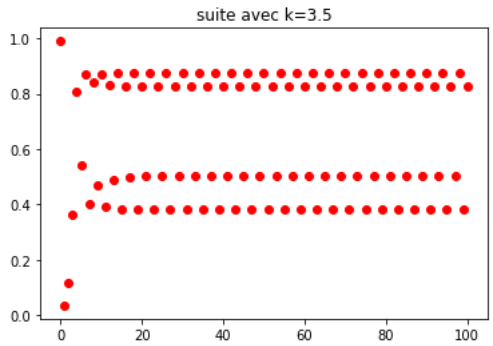



$u_0 = 1$

Les suites se mettent rapidement à osciller entre deux valeurs 0,75 et 0,55. La population d'arbres occupe, une année sur deux, les trois quarts de la population maximale, et l'autre année, environ la moitié de la population maximale.

- **$k = 3{,}5$**




Les suites se mettent rapidement à osciller entre quatre valeurs plus difficiles à lire. La population d'arbres s'élève à la même proportion de la population maximale de manière périodique une année sur quatre.

Pour trouver ces valeurs, il suffit de calculer quatre termes consécutifs de la suite à partir d'un rang assez grand.

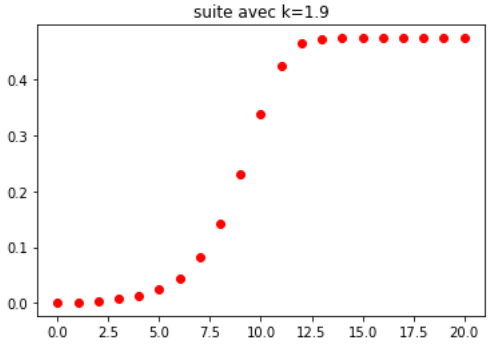
PnsurPmax (3.5,100,0.1) = 0.8269407065914387

PnsurPmax (3.5,101,0.1) = 0.5008842103072179

PnsurPmax (3.5,102,0.1) = 0.8749972636024641

PnsurPmax (3.5,103,0.1) = 0.38281968301732416

- $k = 1{,}9$ et $u_0 = 0{,}001$



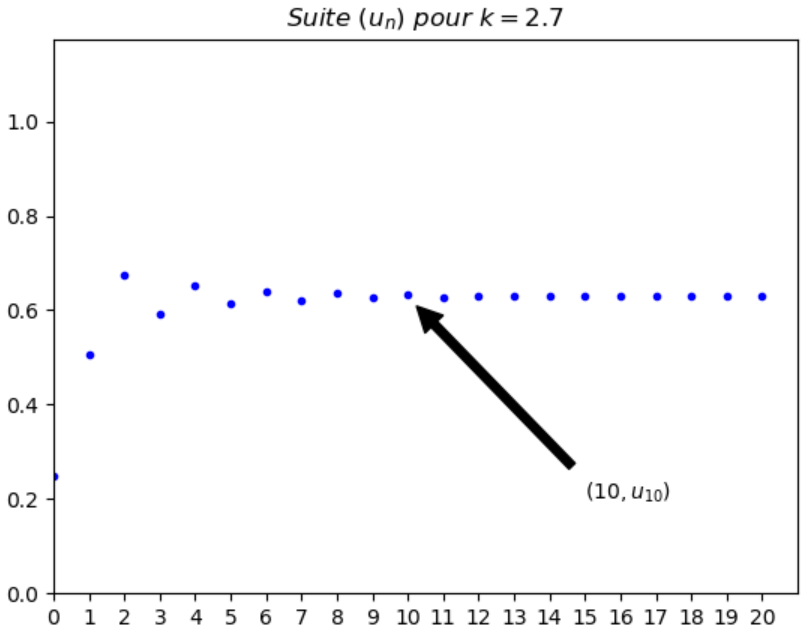

La suite semble croissante puis constante. Mais est-ce bien le cas ?

Il faut saisir pour $u_{20}$ « print(PnsurPmax(1.9,20,0.001)) » et on trouve 0.4736842104128777, pour $u_{21}$ « print(PnsurPmax (1.9,21,0.001)) » et on trouve 0.4736842105149719 .

La suite n'est pas constante, elle semble encore croissante mais très lentement.

Pour aller plus loin sur l'affichage, on pourra considérer la syntaxe suivante :

```
from matplotlib.pyplot import*

def termes(k,n,u):
    """Renvoie la liste des termes de la suite de 0 à n pour une valeur du
    paramètre k et de premier terme u.
    """
    valeurs=[u] # la liste valeurs est initialisée avec le premier terme de la suite
    for compteur in range(n):
        u=k*u*(1-u)
        valeurs.append(u)
    return valeurs


def trace(liste_x,liste_y,k):
    """Représente le nuage de points dont les abscisses sont données
    par la liste liste_x et les ordonnées sont données par la liste liste_y
    """
    xlim(0,len(liste_x)) # les bornes de l'axe des abscisses
    ylim(0,max(liste_y)+0.5) # les bornes de l'axe des ordonnées
    xticks(liste_x) # les marques sur l'axe des abscisses
    annotate(r'$\left(10,u_{10}\right)$',xy=(10,liste_y[11]),xytext=(15,0.2),
                arrowprops=dict(facecolor='black',shrink=0.05))# annotation
    plot(liste_x,liste_y,'b.')# b pour bleu et . pour un point
    title(r'$Suite\ (u_n)\ pour\ k=%s$'%k)# formatage de la chaine à afficher en titre
    # utilisation des expressions latex. Le r devant la chaîne indique qu'elle doit être
    # interprétée de façon brute sinon il faut utiliser des caractères d'échappement.
    show()

k=2.7
u=0.25
trace(range(21),termes(k,20,u),k)
```

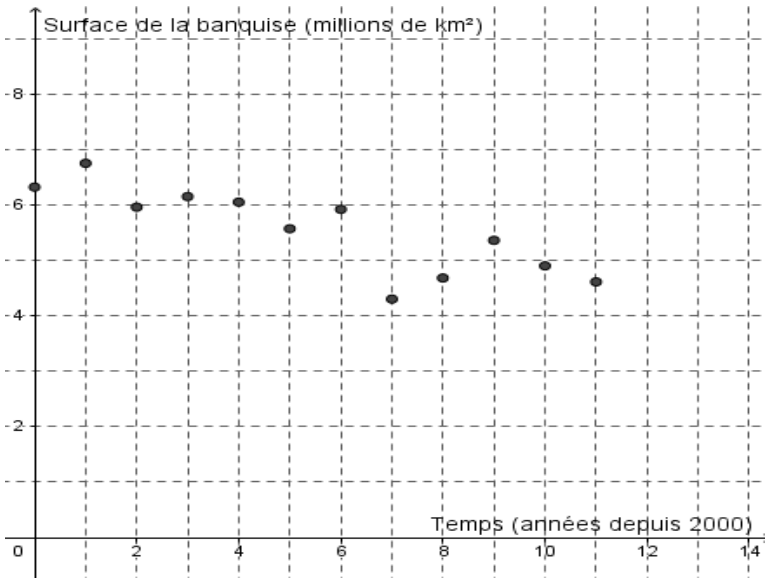



## 2.2. Etudier des phénomènes continus

### Une première situation d'ajustement

Le tableau donne les surfaces minimales $(y_i)$ de la banquise, en millions de km², pour les années $(x_i)$ que l'on représente par un nuage de points $(x_i, y_i)$

| Année | Rang ($x_i$) | Superficie de la banquise ($y_i$) |
|---|---|---|
| 2000 | 0 | 6,32 |
| 2001 | 1 | 6,75 |
| 2002 | 2 | 5,96 |
| 2003 | 3 | 6,15 |
| 2004 | 4 | 6,05 |
| 2005 | 5 | 5,57 |
| 2006 | 6 | 5,92 |
| 2007 | 7 | 4,3 |
| 2008 | 8 | 4,68 |
| 2009 | 9 | 5,36 |
| 2010 | 10 | 4,9 |
| 2011 | 11 | 4,61 |




L'objectif est de déterminer la meilleure relation affine possible entre la surface de la banquise $y$ et le temps $x$, en années après 2000.

**Phase 1 : Au jugé**

L'approche doit être progressive en commençant par le tracé d'une droite passant par deux points du nuage et qui semble la plus proche possible du nuage. On détermine l'équation réduite de la droite, par le calcul et par un logiciel. Le calcul, par exemple, de la moyenne des coefficients directeurs des droites tracées permet d'avoir une valeur de référence pour la droite réalisant un ajustement affine. C'est l'occasion de revenir sur les équations de droites.

**Phase 2 : Une droite possible, la droite de Mayer**

Le nuage de points est divisé en deux sous-nuages de 6 points pour lesquels le calcul des coordonnées des points moyens $G_1$ et $G_2$ de chaque sous-nuage permet de tracer $(G_1G_2)$ et de la comparer à la droite tracée au jugé .

Une fois les coordonnées des points $G_1$ et $G_2$ réalisés, la droite et son équation s'obtiennent grâce à la commande à rédiger dans la ligne de saisie : **f(x)=Droite(G_1,G_2) .**

**Phase 3 : Recherche au jugé de la meilleure droite d'ajustement**

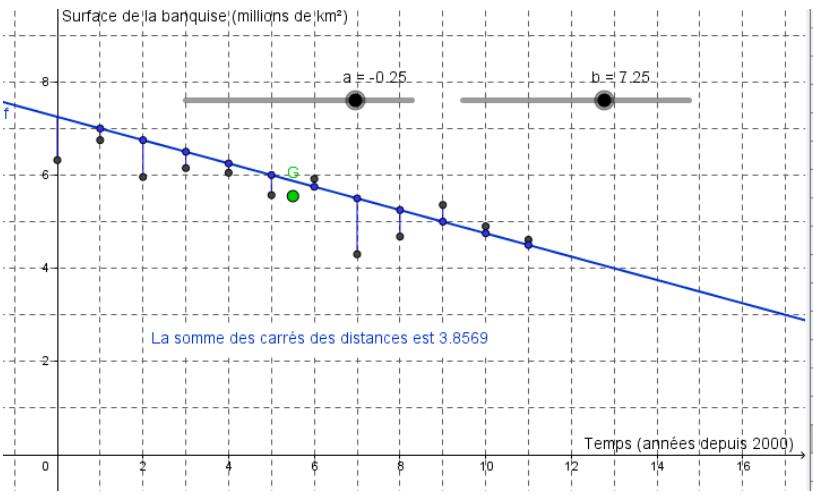
Dans cette partie, on détermine la somme des carrés des distances des points du nuage à la droite tracée « au jugé » à l'aide d'un logiciel.

Les points de la droite de même abscisse que les points du nuage permettent de dessiner et mesurer les distances des points du nuage à la droite. Le calcul de la somme des carrés des distances des points du nuage à la droite permet de constater la diversité des résultats qu'il est intéressant à ce stade de comparer.

Avec Geogebra, ces distances sont visualisées pour différentes positions de droites.

Dans un fichier où les valeurs ont été entrées dans les cellules A1 à B12 du tableur et où les points de coordonnées $(A_i, B_i)$ ont été créés dans la colonne C (**=(A1,B1)** et « tirer vers le bas » pour dupliquer la formule), on peut :

- créer un curseur **a** (coefficient directeur de la droite),
- créer un curseur **b** (ordonnée à l'origine de la droite),
- créer la fonction définie par $f(x) = a*x+b$ ,
- dans la colonne E, créer la liste des points de la droite de même abscisse que les points du nuage, de coordonnées $(A_i, f(A_i))$ **(=(A1,f(A1))**),
- représenter les points de la droite de même abscisse que les points du nuage, de coordonnées $(A_i, f(A_i))$ grâce à une sélection de la colonne E, clic droit, ***Créer – Liste de points*** **,**
- créer dans la colonne F les segments ***Point du nuage-Point de la droite de même abscisse*** **(=Segment(C1,E1) ),**
- dans la colonne G, calculer les carrés des longueurs des segments (**=F1^2**),
- dans la cellule G14, afficher la somme des carrés des longueurs des segments (**=somme(G1 :G12)**).



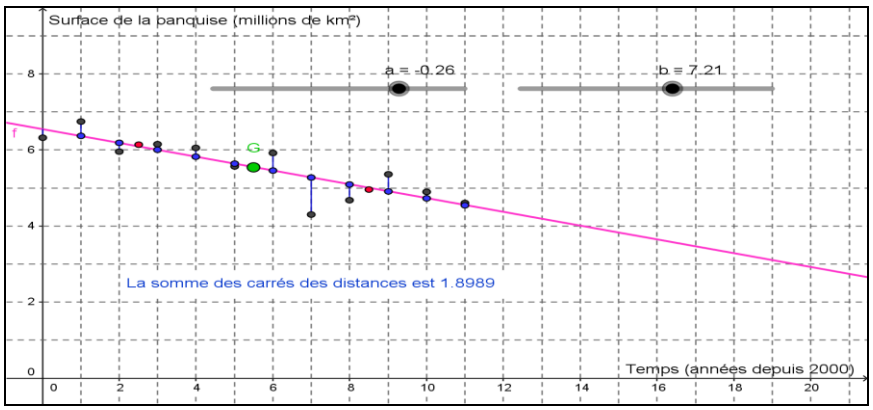

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 6.32 | (0, 6.32) | 7.25 | (0, 7.... | 0.93 | 0.8649 |
| 2 | 1 | 6.75 | (1, 6.75) | 7 | (1, 7) | 0.25 | 0.0625 |
| 3 | 2 | 5.96 | (2, 5.96) | 6.75 | (2, 6.... | 0.79 | 0.6241 |
| 4 | 3 | 6.15 | (3, 6.15) | 6.5 | (3, 6.5) | 0.35 | 0.1225 |
| 5 | 4 | 6.05 | (4, 6.05) | 6.25 | (4, 6.... | 0.2 | 0.04 |
| 6 | 5 | 5.57 | (5, 5.57) | 6 | (5, 6) | 0.43 | 0.1849 |
| 7 | 6 | 5.92 | (6, 5.92) | 5.75 | (6, 5.... | 0.17 | 0.0289 |
| 8 | 7 | 4.3 | (7, 4.3) | 5.5 | (7, 5.5) | 1.2 | 1.44 |
| 9 | 8 | 4.68 | (8, 4.68) | 5.25 | (8, 5.... | 0.57 | 0.3249 |
| 10 | 9 | 5.36 | (9, 5.36) | 5 | (9, 5) | 0.36 | 0.1296 |
| 11 | 10 | 4.9 | (10, 4.9) | 4.75 | (10, 4... | 0.15 | 0.0225 |
| 12 | 11 | 4.61 | (11, 4.... | 4.5 | (11, 4... | 0.11 | 0.0121 |
| 13 | | | | | | | |
| 14 | | | | | | | 3.8569 |
| 15 | | | | | | | |

En faisant bouger les curseurs, la somme des carrés varie.

**Phase 4 : La droite d'ajustement par la méthode des moindres carrés**

La droite d'ajustement affine proposée par Geogebra, s'obtient en saisissant **=Ajustlin[C1 :C12]** dans la ligne de saisie.

Reprendre ce qui a été fait à la phase 3 en remplaçant la droite obtenue au jugée par la droite d'ajustement.



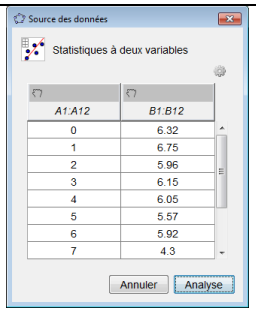
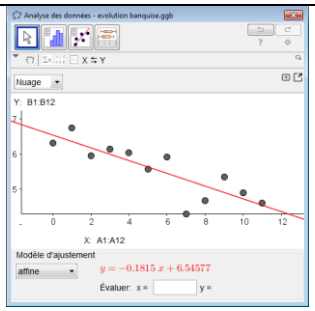

C'est l'occasion de remarquer pour cette droite que la somme des carrés des distances est minimale et qu'elle passe par le point moyen du nuage de points $(x_i ; y_i)$.

Sur le graphique on pourra :

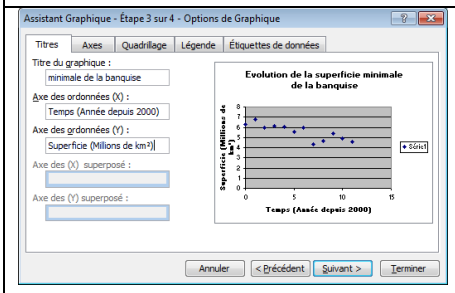
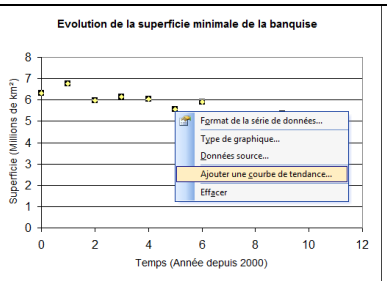
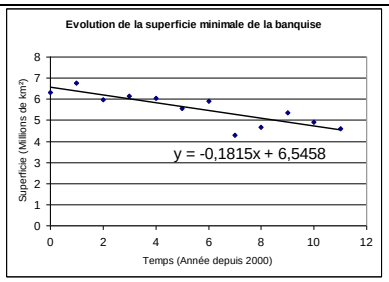
- Afficher la somme des carrés des écarts obtenue par un tracé au jugé (valeurs de a et b variables)
- Afficher la somme des carrés des écarts obtenue par la méthode des moindres carrés
- Comparer ces deux valeurs, en faisant varier a et b. C'est l'occasion de faire remarquer aux élèves que la méthode des moindres carrés est la plus satisfaisante.

D'autres outils numériques permettent de déterminer l'équation de la droite d'ajustement affine d'une série statistique à deux variables.

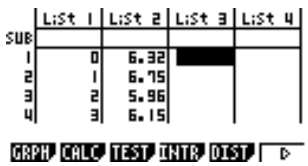
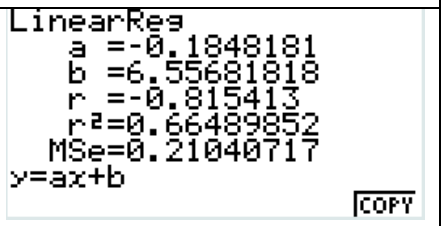
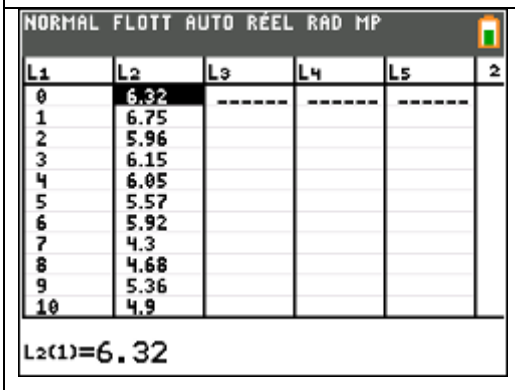
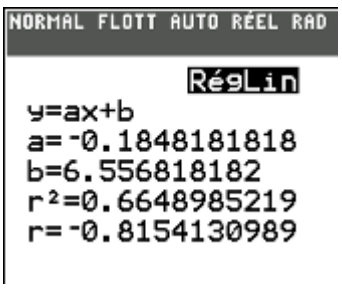
**Geogebra**

| Sélectionner les données<br><br>Sélectionner l'outil Statistiques à 2 variables | Sélectionner Analyse : Le nuage de points s'affiche.<br>Sélectionner le modèle d'ajustement affine |
|---|---|
|  |  |

**Tableur**

| | | |
|---|---|---|
| Entrer les données<br>Sélectionner les valeurs<br>Insertion – Graphique – Nuages de points – suivant – suivant<br>A l'étape 3, indiquer les titres du graphique et des axes. | Une fois le nuage de points affiché.<br><br>Cliquer droit sur un des points du nuage<br>Sélectionner Ajouter une courbe de tendance | Sélectionner<br>Type : Linéaire<br>Options : Afficher l'équation |
|  |  |  |

**Calculatrice Menu STATS**

| | | |
|---|---|---|
| **CASIO**<br>Entrer les données<br>Les valeurs $x_i$ dans List1<br>Les valeurs $y_i$ dans List2 | Sélectionner CALC et SET<br>Régler 2Var XList et 2Var YList avec les listes contenant les $x_i$ et les $y_i$ | Revenir à la fenêtre précédente (EXIT)<br>Sélectionner REG – X – ax+b |
|  |  |  |
| **TI**<br>Entrer les données<br>Les valeurs $x_i$ dans List1<br>Les valeurs $y_i$ dans List2 | Revenir au menu STATS,<br>Sélectionner CALC, puis 4 :<br>RégLin(ax+b) , choisir L1 puis L2 et Calculer | |
|  |  |  |

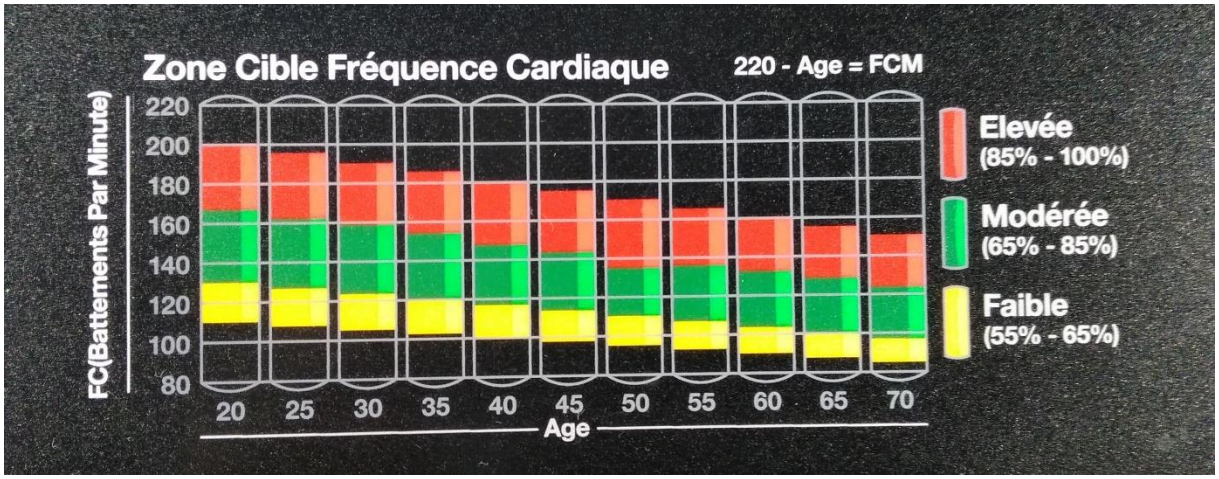
Tous ces outils donnent la même équation de droite pour l'ajustement affine de Y en X.

Il s'agit de la droite pour laquelle la somme des carrés des distances des points du nuage à la droite est minimale. C'est la méthode des moindres carrées.

Cette droite passe par le point moyen du nuage.

## Une seconde situation

L'écran d'un vélo d'appartement propose le diagramme suivant qui permet à l'utilisateur de s'entraîner dans différentes « zones » cardiaques.




Ces différentes zones, symbolisées par des couleurs, correspondent à des intensités différentes en fonction de l'intensité de l'effort et sont basées sur la fréquence cardiaque maximale (FCM en abrégé) donnée en battements par minutes (bpm).

La détermination de cette valeur qui varie en fonction de l'âge de la personne peut se faire de différentes manières.

La première est individualisée et peut se faire à l'aide d'un cardiofréquencemètre lors d'un test de terrain ou dans un centre médical, nous parlerons alors de FCM réelle.

La deuxième mesure est théorique et se déduit à l'aide de formules établies par des études statistiques, nous parlerons alors de FCM théorique.

Une fois cette valeur déterminée, le sportif peut alors s'entraîner dans différentes zones en fonction de ses objectifs.

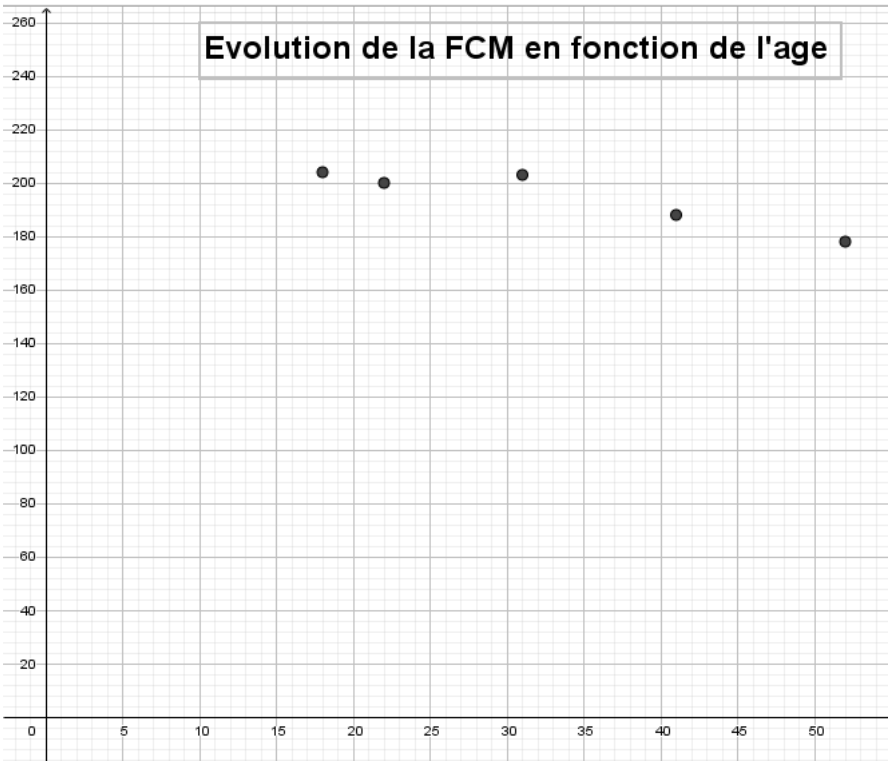
On s'intéresse aux valeurs des FCM pour un groupe de femmes extraites de http://aviron.aix.les.bains.free.fr/doc/fsp.htm (d'autres données plus exhaustives sont disponibles si besoin)

| Age | FCM réelle |
|---|---|
| 18 | 204 |
| 22 | 200 |
| 31 | 203 |
| 41 | 188 |
| 52 | 178 |

Dans un premier temps, le réinvestissement de la notion d'équation de droite peut se faire, pour chaque couple de valeurs, par le tracé de la droite et la détermination de l'équation réduite. C'est un travail collaboratif, avec vérification possible par au moins deux élèves suivant l'effectif de la classe. On reprend ainsi des notions vues en classe de seconde (équation de droite, programme donnant le coefficient directeur et l'ordonnée à l'origine à partir des coordonnées des points …). Le travail peut être fait à la maison, dans le cas d'un exercice ou d'une situation de mise en place de classe inversée.

Les méthodes d'ajustement peuvent être considérées comme les premières situations de modélisation et s'appuient sur une démarche graphique.

L'utilisation d'outils numériques (Geogebra, tableur-grapheur, …) seront à privilégier pour cette partie.




C'est l'occasion de débattre, en attendant des arguments, sur le choix de la droite qui parait être « la plus proche du nuage de points » et la considérer comme référence.

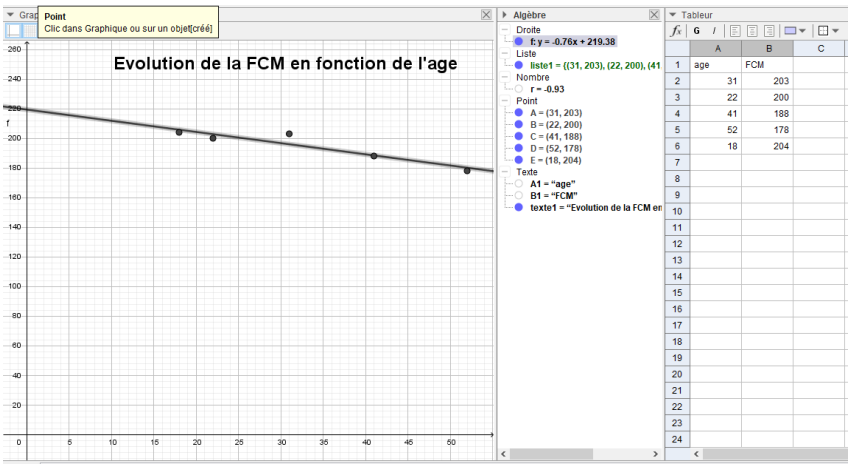
Astrand, un professeur de physiologie suédois, a modélisé la FCM Théorique des femmes en fonction de l'âge par :

$$FCM_{théorique} = 226 - âge$$ [1]

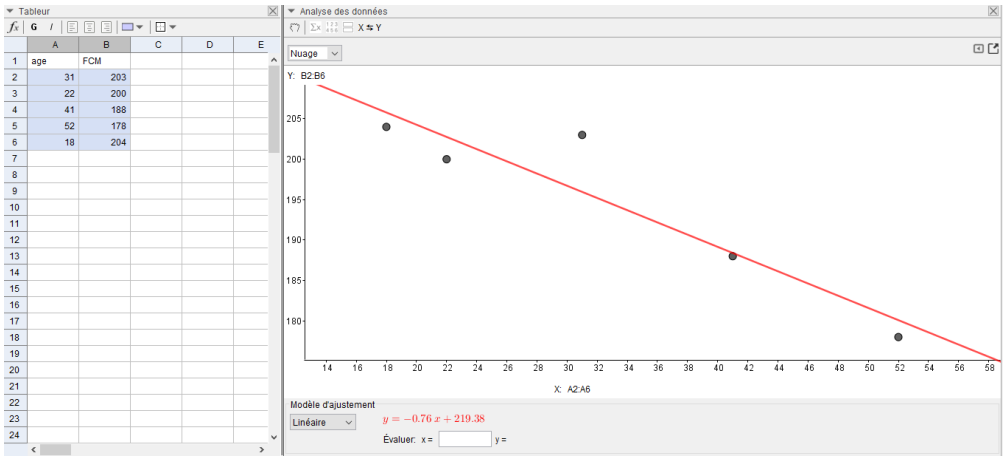
puis tracer cette droite modélisant la $FCM_{théorique}$ sur le nuage de points précédent.

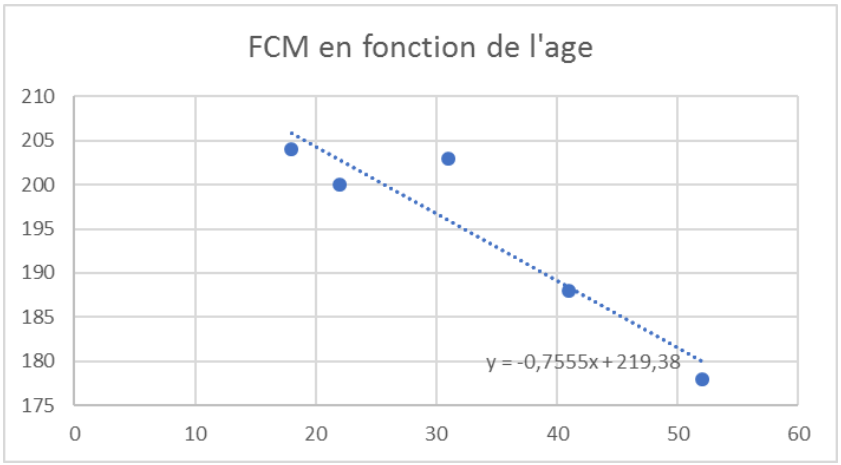
On peut aussi la comparer à la droite d'ajustement que l'on peut déterminer de plusieurs façons :

- En utilisant une liste de points et AjusLin(<listepoints>) en zone de saisie sur Geogebra
- En utilisant l'outil d'Analyse des données de Geogebra
- En utilisant l'ajout de courbe de tendance avec l'affichage de l'équation dans un tableur
- Sur la calculatrice en utilisant le mode statistique




[1] Cette formule diffère pour les hommes et devient $FCM_{théorique} = 220 - âge$.

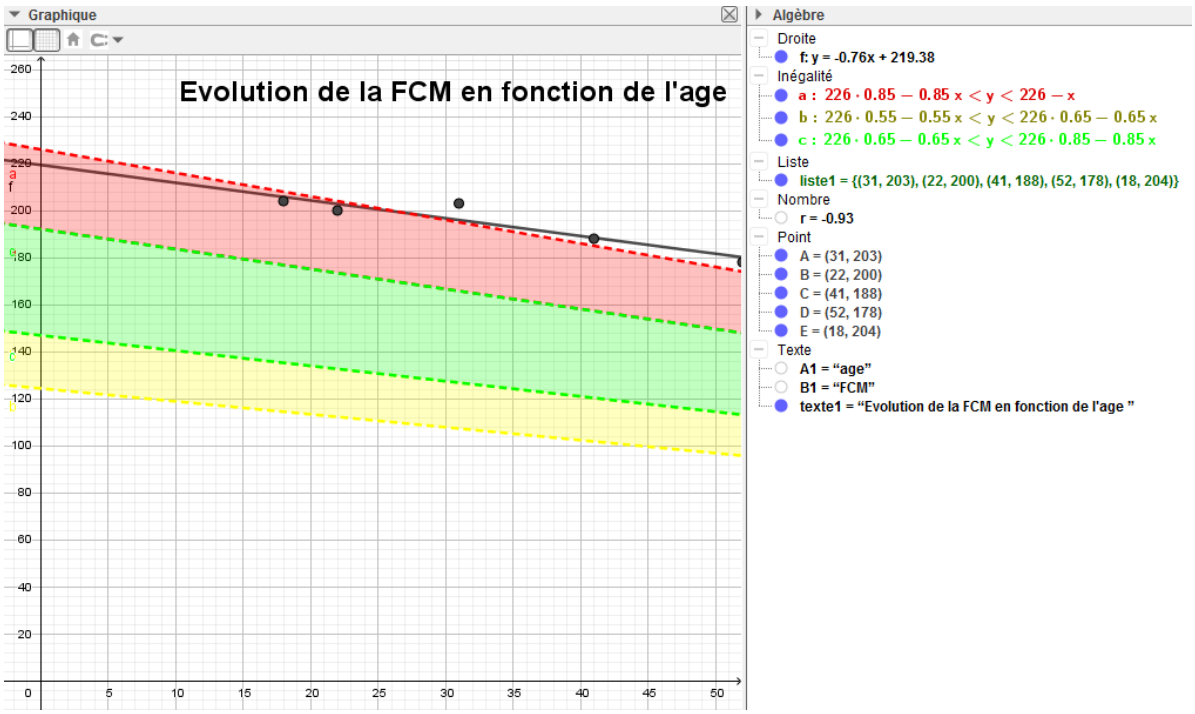



En reprenant la formule d'Astrand, on considère trois zones d'intensité différentes :

- Zone Jaune entre 55% et 65% % de la FCM, ce qui correspond à un entraînement de base en endurance.
- Zone verte entre 65% et 85% de la FCM, ce qui correspond à un entraînement en endurance active.
- Zone rouge entre 85 % et 100% de la FCM, ce qui correspond à un entraînement très intensif.

On peut alors représenter les trois zones sur le graphique précédent à partir de la droite de référence choisie par la classe.

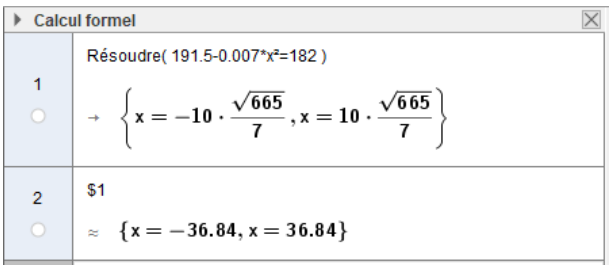


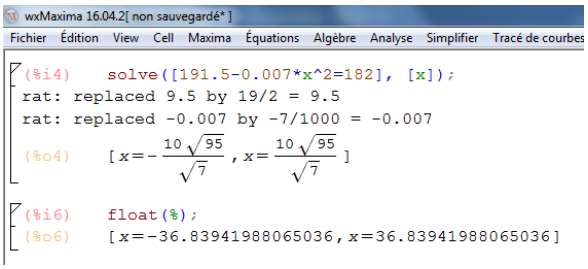
Il existe également une modélisation de la FCM théorique en fonction de l'âge par une formule qui semble plus valide car elle épouse mieux la tendance de l'évolution globale de la FCM en fonction de l'âge et ne prend pas en compte le fait d'être un homme ou une femme :

$$FCM = 191{,}5 - 0{,}007 \times âge^2$$

On peut alors faire de nouveaux calculs de FCM, et surtout déterminer l'âge à partir duquel il ne faut pas dépasser une certaine FCM, 182 battements par minutes par exemple.

L'utilisation d'un solveur d'équation comme celui de la calculatrice (ou de Geogebra ou de wxMaxima) permet de répondre à cette dernière question, à condition de faire un travail sur les fonctions trinômes pour utiliser le résultat d'égalité pour l'inéquation posée :




En TIM, la réalisation d'une feuille de calcul permettant d'automatiser le calcul de la FCM théorique et des zones cibles en fonction de l'âge permettra de mettre en œuvre les principales fonctions d'un tableur-grapheur.

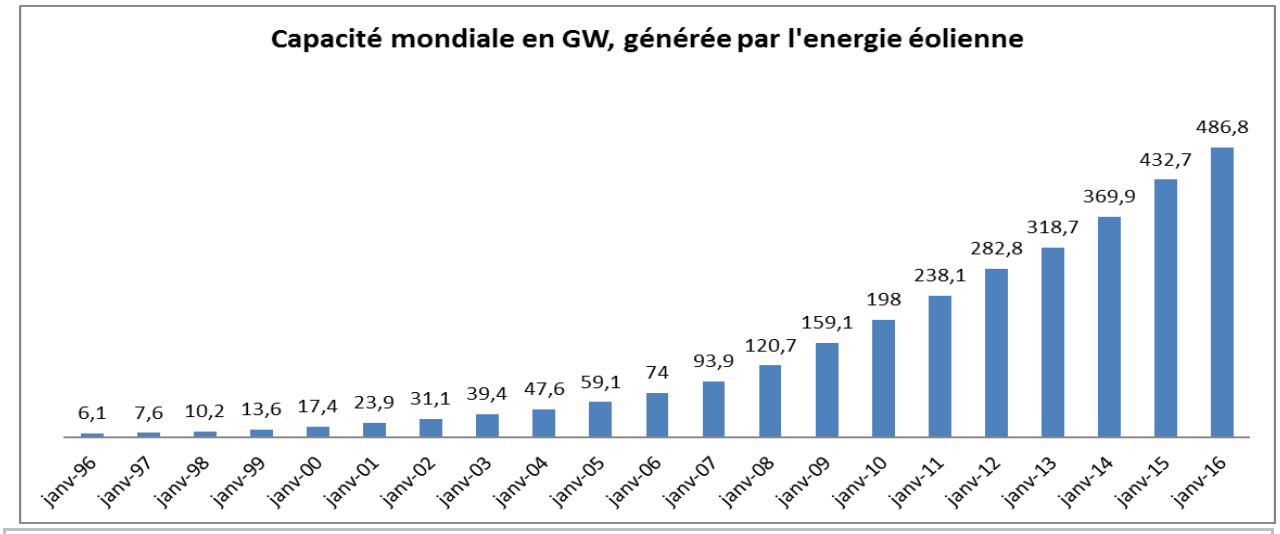
Il est également possible de mettre en œuvre un travail collaboratif avec le professeur d'EPS afin de travailler avec les données « réelles » obtenues lors de tests de terrain.

## Des suites aux fonctions exponentielles

Comme cela a pu être fait en classe de seconde pour montrer que les liens entre les grandeurs n'étaient pas uniquement linéaires et affines, justifiant ainsi l'étude d'autres fonctions, la situation suivante donne à voir un phénomène discret dont l'évolution n'est pas constante.

L'évolution de la capacité éolienne mondiale en GW entre 1996 et 2016 est donnée dans le tableau suivant (source GWEC)

| Année | 01/1996 | 01/1997 | 01/1998 | 01/1999 | 01/2000 | 01/2001 | 01/2002 | 01/2003 | 01/2004 | 01/2005 | 01/2006 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Capacité en GW | 6,1 | 7,6 | 10,2 | 13,6 | 17,4 | 23,9 | 31,1 | 39,4 | 47,6 | 59,1 | 74,0 |
| Année | 01/2007 | 01/2008 | 01/2009 | 01/2010 | 01/2011 | 01/2012 | 01/2013 | 01/2014 | 01/2015 | 01/2016 | |
| Capacité en GW | 93,9 | 120,7 | 159,1 | 198,0 | 238,1 | 282,8 | 318,7 | 369,9 | 432,7 | 486,8 | |



Il importe de montrer (graphiquement, par le calcul, …) dans un premier temps que la modélisation par une suite arithmétique n'est pas adaptée dans ce cas. En lien avec les « automatismes », la détermination des taux d'évolution annuels en est une illustration.

**<u>Etape 1</u> : Un autre modèle de suite que les suites arithmétiques**

Le calcul des taux d'évolution entre chaque année donne :

(*on pourra à cette occasion prévoir un travail de groupe pendant lequel on répartira les calculs des taux.)*

| 96-97 | 97-98 | 98-99 | 99-00 | 00-01 | 01-02 | 02-03 | 03-04 | 04-05 | 05-06 |
|---|---|---|---|---|---|---|---|---|---|
| 0,246 | 0,342 | 0,333 | 0,279 | 0,374 | 0,301 | 0,267 | 0,208 | 0,242 | 0,252 |

| 06-07 | 07-08 | 08-09 | 09-10 | 10-11 | 11-12 | 12-13 | 13-14 | 14-15 | 15-16 |
|---|---|---|---|---|---|---|---|---|---|
| 0,269 | 0,285 | 0,318 | 0,245 | 0,203 | 0,188 | 0,127 | 0,161 | 0,170 | 0,125 |

Les taux d'évolutions semblent assez proches. Comme valeur moyenne du taux d'évolution, on peut utiliser la valeur de la moyenne arithmétique des taux, même si ce n'est pas juste mathématiquement, est une valeur acceptable, assez proche de la valeur réelle car elle est proche de 0. La valeur théorique du taux moyen est la moyenne géométrique de ces taux d'évolution mais cette notion n'est pas connue pour l'instant.

La moyenne arithmétique des taux annuels d'évolution est de 0,247 (la valeur de la moyenne géométrique est 0,236).

En lien avec le coefficient multiplicateur, $u_{n+1}$ s'exprime en fonction de $u_n$ ce qui permet de définir la notion de suite géométrique

Le modèle de suite $(u_n)$ géométrique se justifie, avec comme raison $q = 1,247$ et premier terme $u_0 = 6,1$ et telle que $u_n$ désigne la capacité cumulée dans le monde en GW l'année 2016 + *n* .

Valeurs obtenues par la modélisation :

*on pourra à cette occasion prévoir un travail de groupe pour lequel on répartira les calculs des valeurs avec ce modèle, ainsi qu'une représentation graphique pour comparer les valeurs réelles et valeurs du modèle.*

| Année | 01/1996 | 01/1997 | 01/1998 | 01/1999 | 01/2000 | 01/2001 | 01/2002 | 01/2003 | 01/2004 | 01/2005 | 01/2006 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Capacité en GW | 6,1 | 7,6 | 9,5 | 11,8 | 14,6 | 18,4 | 22,9 | 28,6 | 35,7 | 44,5 | 55,5 |

| Année | 01/2007 | 01/2008 | 01/2009 | 01/2010 | 01/2011 | 01/2012 | 01/2013 | 01/2014 | 01/2015 | 01/2016 |
|---|---|---|---|---|---|---|---|---|---|---|
| Capacité en GW | 69,2 | 86,2 | 107,5 | 134,1 | 167,2 | 208,5 | 260,1 | 324,3 | 404,4 | 504,3 |

**<u>Etape 2</u> : D'un modèle discret à un modèle continu**

Les suites géométriques ayant été abordées en première, cela peut servir d'appui en terminale avant d'introduire les fonctions exponentielles. La suite $u_n = 6,1 \times 1,247^n = f(n)$ .

A partir de la même situation, la valeur obtenue au bout de 6 mois s'obtient en respectant le fait que l'évolution sur les 6 premiers mois de l'année est la même que sur les 6 mois suivants.

| Année | 01/1996 | 07/1997 | 01/1997 |
|---|---|---|---|
| Capacité cumulée en GW | 6,1 | *x* | 7,6 |

Les deux taux sont égaux : $\frac{x-6,1}{6,1}=\frac{7,6-x}{x}$.

Plusieurs approches sont possibles :

Le solveur de la calculatrice permet de déterminer une valeur approchée de cette équation :

résoudre( $\frac{X-6.1}{6.1}-\frac{7.6-X}{X}$ ,X,6.1)

6.808817812

Pour la résolution algébrique : $\frac{x-6,1}{6,1}=\frac{7,6-x}{x} \Leftrightarrow x^2-6,1x=6,1\times7,6-6,1x \Leftrightarrow x^2=6,1\times7,6$.

Soit $x=\sqrt{6,1\times7,6}$ soit 6,8 GW environ. Cela donne du sens à $f(0,5)=6,1\times1,247^{0,5}$.

$f(0,5)=\sqrt{u_0u_1}=\sqrt{f(0)f(1)}=6,1\times1,247^{0,5}$.

*On pourra à cette occasion prévoir un travail de groupe pour lequel on répartira les calculs des valeurs intermédiaires. On tracera alors le nuage de points en rajoutant les nouveaux points. Chaque groupe pourra placer son point au tableau dans un fichier tableur et si le point semble incohérent, un calcul commun peut alors être réalisé pour correction.*

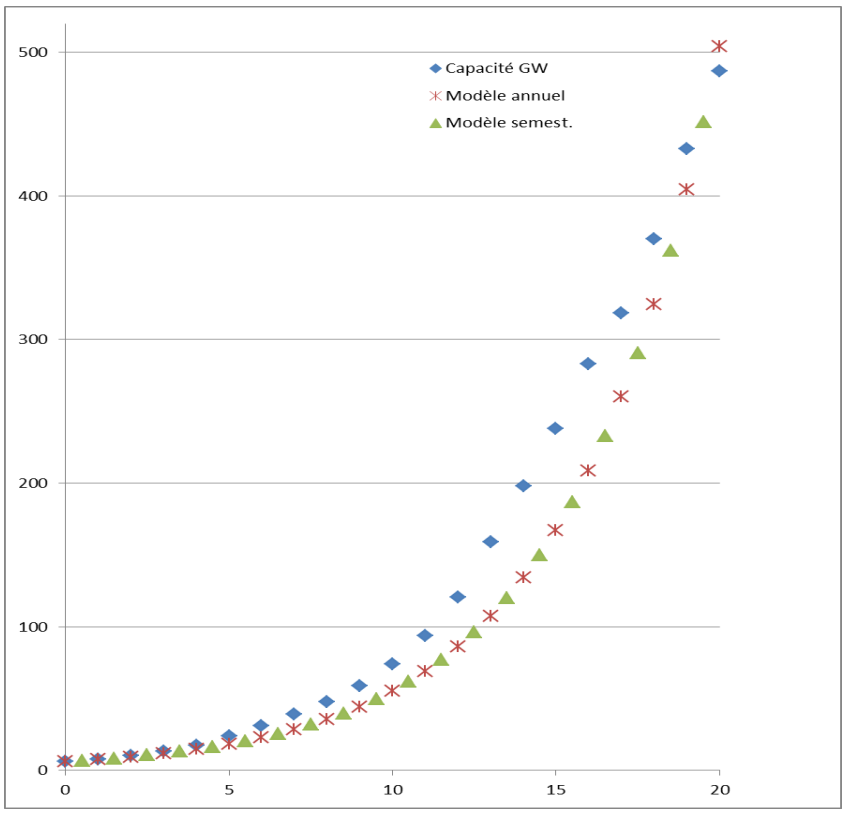
**<u>Etape 3</u> On représente pour comprendre**

Le nuage de points avec un pas de temps annuel avait pour abscisses des nombres entiers et les images sont $f(t)=6,1q^t$. Avec le pas de temps semestriel, on obtient des abscisses décimales 0, 0.5, 1, 1,5 etc…

Ainsi $f(0,5)=6,1q^{0,5}=\sqrt{6,1\times7,6}$

$f(1,5)=\sqrt{f(1)f(2)}=\sqrt{6,1q\times6,1q^2}=6,1q\sqrt{q}=6,1q\times q^{0,5}$

On obtient le graphique sur tableur, mais qui peut se faire avec Geogebra :



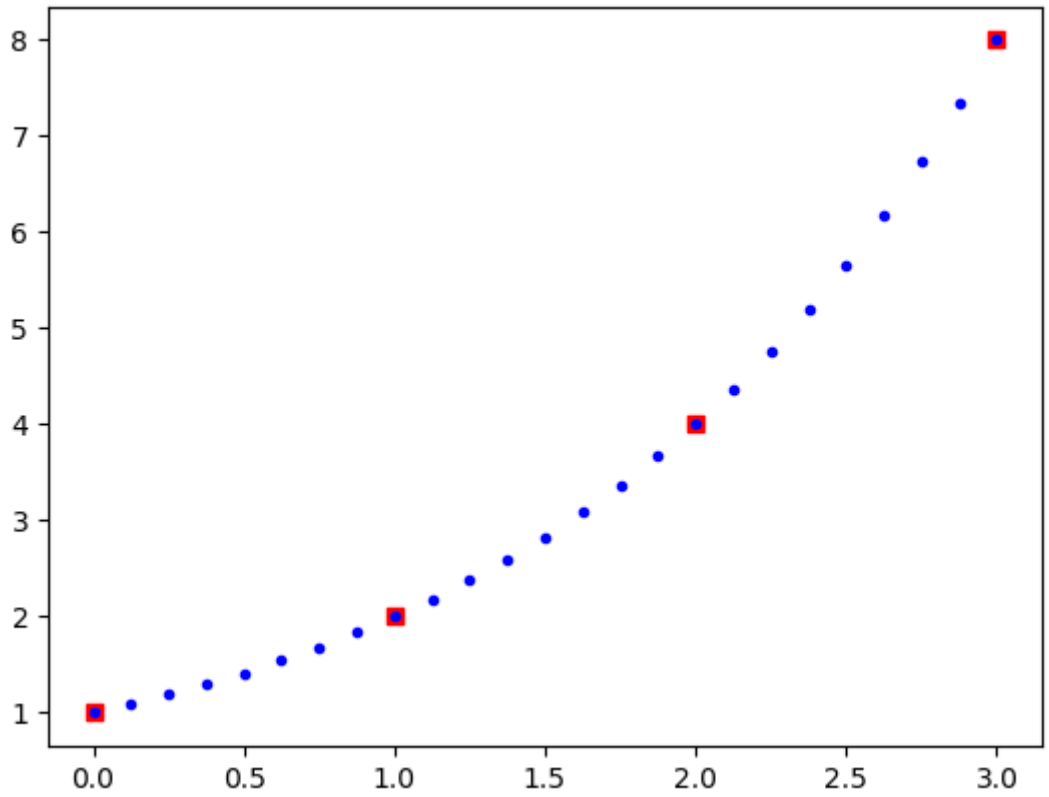

En réitérant avec un pas de temps trimestriel, on est amené à calculer $q^{0,25}, q^{0,75},...$

On peut donc étendre à la fonction $x \mapsto q^x$, dont on obtient les variations à partir de celles des suites géométriques de raison $q > 0$.

On peut programmer en Python un algorithme qui permet de déterminer les moyennes géométriques entre deux valeurs pour un nombre de valeurs données, avec plusieurs itérations (ici 3):

On importe uniquement la fonction « sqrt » (pour square root i.e. racine carrée) du module « math » :

```
from math import sqrt
from matplotlib.pylab import *
############################################################
def moyenneGeo(a,b):
    """
    Retourne la moyenne géométrique des réels a et b positifs.
    """
    return sqrt(a*b)
############################################################
def moyenneArith(a,b):
    """
    Retourne la moyenne arithmétique des réels a et b
    """
    return (a+b)/2
############################################################
def insertAbscisse(liste):
    """
    Insère la moyenne arithmétique entre deux éléments de la liste
    Retourne la liste construite
    """
    listeReturn=[]
    listeReturn.append(liste[0])
    for i in range(1,len(liste)):
        listeReturn.append(moyenneArith(liste[i-1],liste[i]))
        listeReturn.append(liste[i])
    return listeReturn
############################################################
def insertOrdonnee(liste):
    """"
    Insère la moyenne géométrique entre deux éléments de la liste
   """
    listeReturn=[]
    listeReturn.append(liste[0])
    for i in range(1,len(liste)):
        listeReturn.append(moyenneGeo(liste[i-1],liste[i]))
        listeReturn.append(liste[i])
    return listeReturn
############################################################
abscisse=[i for i in range(4)]# On peut modifier la valeur 4.
ordonnee=[2**i for i in abscisse]# On peut le faire d'autres valeurs que 2.
                                 # Avec 1.247 de l'exemple
plot(abscisse,ordonnee,"rs")
########## On itère 3 fois par exemple ############
for i in range(3):
    abscisse=insertAbscisse(abscisse) # A éviter mais ici ça marche.
    ordonnee=insertOrdonnee(ordonnee)# idem

plot(abscisse, ordonnee,"b.")
show()
```




## Des fonctions exponentielles à la fonction exponentielle

La fonction exponentielle $x \mapsto e^x$ est une fonction de la forme $x \mapsto a^x$.

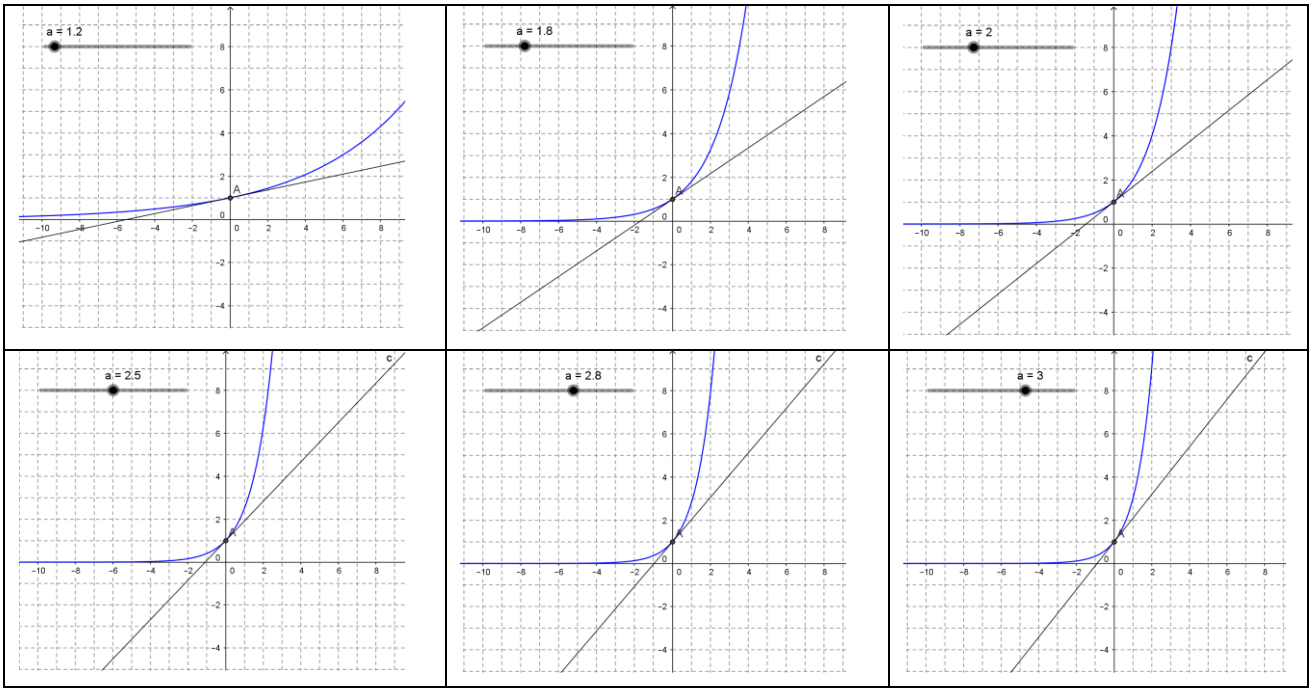
Le nombre $e$ est défini comme le réel $a$ pour lequel la tangente au point d'abscisse 0 à la courbe de la fonction $x \mapsto a^x$ a une pente de 1.

En effet, si $\lim\limits_{h \to 0} \dfrac{e^h - 1}{h} = 1$, alors pour tout réel *x*, $\lim\limits_{h \to 0} \dfrac{e^{x+h} - e^x}{h} = \lim\limits_{h \to 0} e^x \dfrac{e^h - 1}{h} = e^x$, ce qui signifie que cette fonction exponentielle a pour dérivée elle-même.

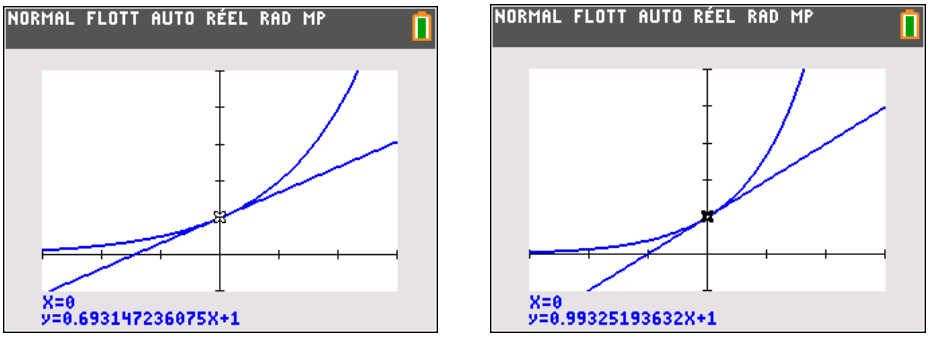
Cela revient à chercher une valeur approchée du nombre $e$ de manière de plus en plus précise. Ce peut être l'objet d'un problème ouvert de recherche en groupe.

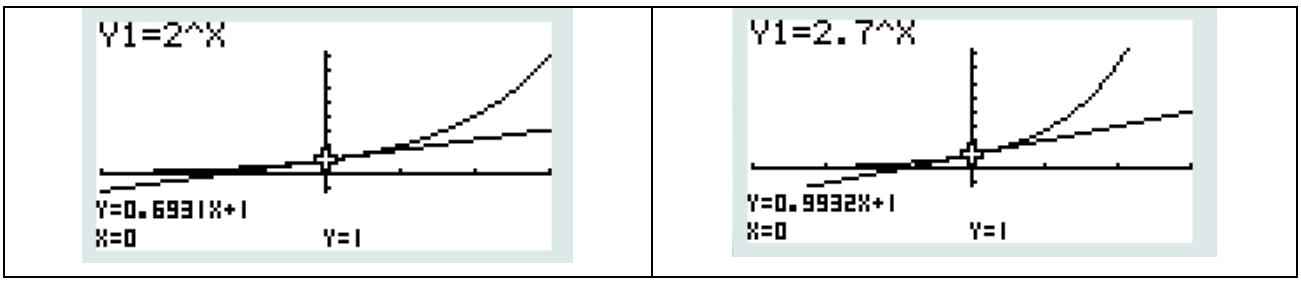
Dans un premier temps, on cherche à déterminer la pente de la tangente à la courbe de plusieurs fonctions $x \mapsto a^x$ pour avoir un encadrement de $e$.

On pourra sur Geogebra ou sur calculatrice donner les courbes de plusieurs fonctions de la forme $x \to a^x$ avec leur tangente au point d'abscisse 0 pour déterminer une valeur arrondie au dixième.

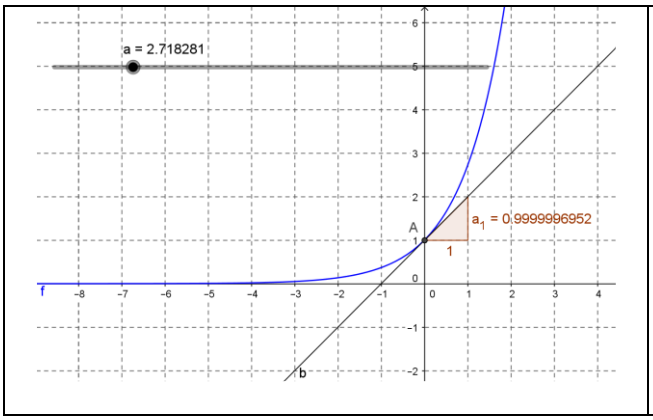


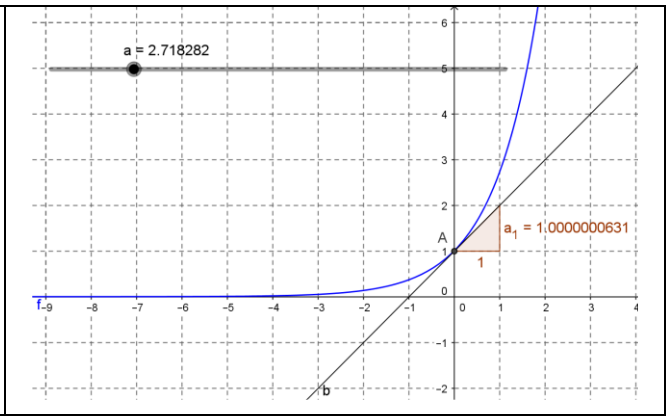

ou utiliser la fonction permettant de tracer la tangente sur calculatrice (ici pour a=2, puis a=2,7):




On affine l'encadrement du nombre $e$ à l'aide d'un logiciel de géométrie dynamique (mais on peut aussi le faire à la calculatrice).




On obtient $2{,}718281 < e < 2{,}718282$. Difficile d'aller plus loin avec Geogebra.

Pour pousser plus loin la précision de la valeur approchée du nombre $e$, on peut programmer un algorithme, en ayant fait constater à l'aide de Geogebra que si $a$ augmente, le coefficient directeur de la tangente en 0 à la courbe représentative de $x \mapsto a^x$ augmente.

| | | |
|---|---|---|
| $a \leftarrow 2$<br>$h \leftarrow 10^{-8}$<br>$d \leftarrow 1$<br>Lire $p$<br>Tant que $\frac{a^h - 1}{h} - 1 > p$<br>Si $\frac{a^h - 1}{h} < 1$ alors<br>$a \leftarrow a + d$<br>Sinon<br>$a \leftarrow a - d$<br>$d \leftarrow d / 2$<br>Afficher $a$ | Avec Python, on définit une fonction appoxim_e(p) où p est la précision souhaitée<br>`def approxim_e(p):`<br>`  a=2`<br>`  d=1`<br>`  h=10**-8`<br>`  while abs((a**h-1)/h-1)>p:`<br>`  if ((a**h-1)/h)<1:`<br>`  a=a+d`<br>`  else:`<br>`  a=a-d`<br>`  d=d/2`<br>`  return(a)` | En appelant cette fonction avec différentes valeurs de p, on obtient ces valeurs approchées de e<br>`>>> from math import*`<br>`>>> exp(1)`<br>`2.718281828459045`<br>`>>> approxim_e(0.01)`<br>`2.71875`<br>`>>> approxim_e(0.0001)`<br>`2.71826171875`<br>`>>> approxim_e(0.000001)`<br>`2.718280792236328` |

## Une approche de la fonction logarithme décimal

La fonction logarithme décimal comme la réciproque de la fonction $x \mapsto 10^x$ définie pour tous les réels.

Cela permet déjà d'établir que l'ensemble de définition de cette fonction est $]0;+\infty[$ puisque pour tout réel $x, 10^x > 0$.

Les premières valeurs donnent :

$\log(1) = 0$ car $10^0 = 1$, $\log(10) = 1$ car $10 = 10^1$, $\log(100) = 2$ car $10^2 = 100$ et ainsi de suite.

Mais comment donner alors du sens à $\log(5)$ ?

Cela revient à chercher la valeur du réel $x$ tel que $10^x = 5$. Le fait que 1 < 5 < 10 donne que $0 < \log(5) < 1$ puisque $x \mapsto 10^x$ est une fonction strictement croissante.

Le solveur de la calculatrice permet d'en avoir une valeur approchée plus précise.

Il y a même une fonction **solve** ou **résoudre** (0 correspond à une valeur de départ qui initialise la recherche).

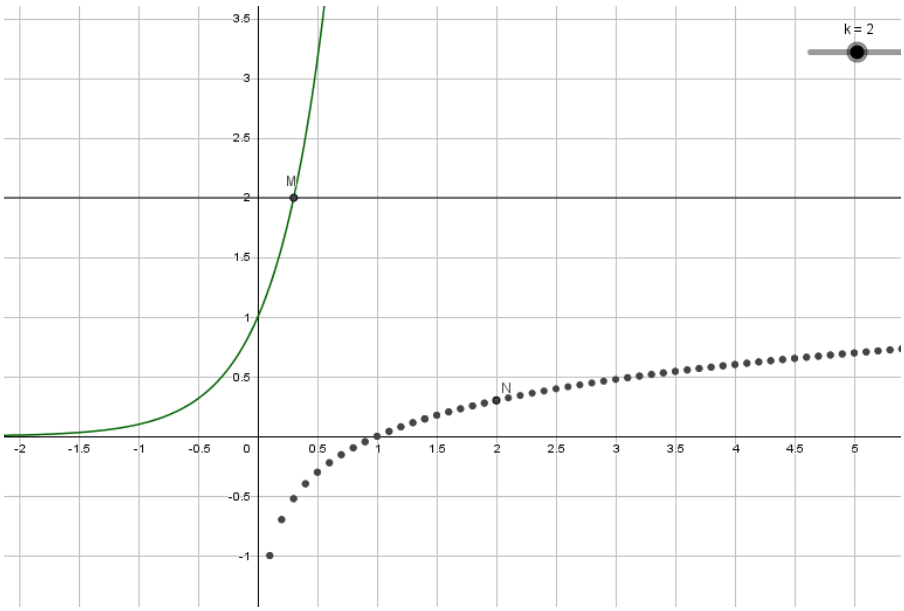
| TI | CASIO |
|---|---|
| résoudre(10^X−5,X,0)<br>0.6989700043 | Solve(10^X−5=0,X)<br>0.6989700043 |

On peut même automatiser cela dans un programme par exemple qui permet de donner les valeurs de log(I) pour différentes valeurs de I, ici de 1 à 10.

| TI | CASIO |
|---|---|
| PROGRAM:RES<br>:For(I,1,10)<br>:résoudre(10^X−I,X,0)→A<br>:Disp I,A<br>:Pause<br>:End | ======RES ======<br>For 1→I To 10<br>Solve(10^X−I=0,X)→A<br>I<br>A<br>Next<br>TOP BTM SRC MENU A⇔a CHAR |

Pour le tracé de la courbe de la fonction $\log$, on peut réaliser une animation Geogebra.

Il suffit de représenter la fonction $x \mapsto 10^x$, de créer un curseur *k* et le point M intersection de la représentation graphique de cette fonction et de *D : y = k.* La création du point $N\ (\ k\ ,\ x(M)\ )$ dont on fait afficher la trace permet de représenter point par point la fonction $\log$. On peut observer la symétrie avec la droite D : y = x.



## Une approche concrète des intégrales

Afin d'évaluer le risque de crue d'un fleuve ou la gestion d'un barrage hydraulique ou un bassin de rétention des eaux pluviales…, il est essentiel de pouvoir déterminer une quantité d'eau à partir d'un débit, ce qui permettra par la suite de déterminer par exemple des hauteurs d'eau par surface.

Ce type de calcul peut se retrouver aussi en médecine pour la mise au point de pompes à perfusion.

Il existe des méthodes (même si elles sont parfois peu fiables) pour évaluer le débit d'un cours d'eau.

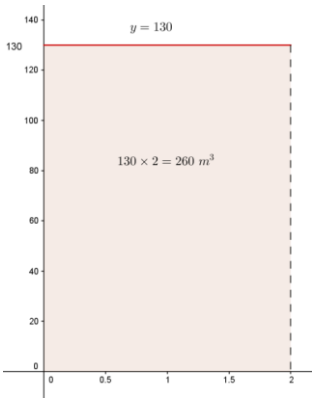
L'objectif de cette approche est d'introduire le calcul intégral à partir d'une problématique accessible aux élèves.

On considère la situation suivante : connaissant le débit à tout instant d'un cours d'eau, ou d'une canalisation, on cherche à évaluer et calculer le volume d'eau s'écoulant en un temps donné.

### 1ère situation : débit constant

Dans un premier temps le débit est considéré comme constant et égal à 130 m³/h pendant 2 heures.

La situation peut être modélisée par la fonction constante $d$ définie sur [0 ; 2] par $d(t)=130$ La détermination de la quantité d'eau s'écoulant revient à calculer l'aire sous la courbe.



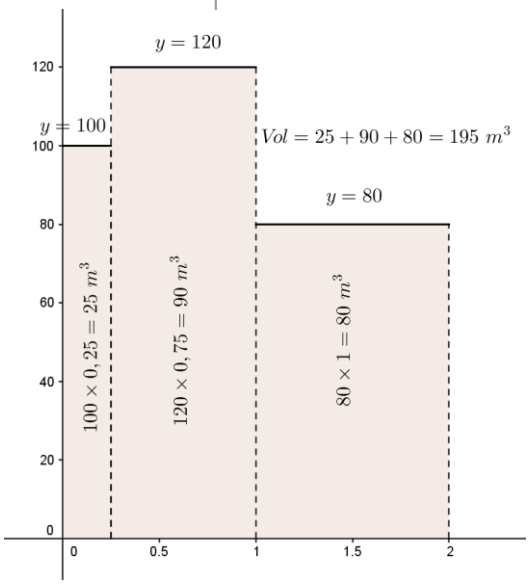

### 2ème situation : débit constant par intervalle de temps

On suppose que le débit est de 100 m³/h pendant 15 minutes, puis 120 m³/h pendant 45 minutes et enfin 80 m³/h pendant une heure.

La situation peut être modélisée par la fonction en escalier

$$d(t)=\begin{cases} 100 \ si \ t\in\left[0;\frac{1}{4}\right[ \\ 120 \ si \ t\in\left[\frac{1}{4};1\right[ \\ 80 \ si \ t\in[1;2[ \end{cases}.$$

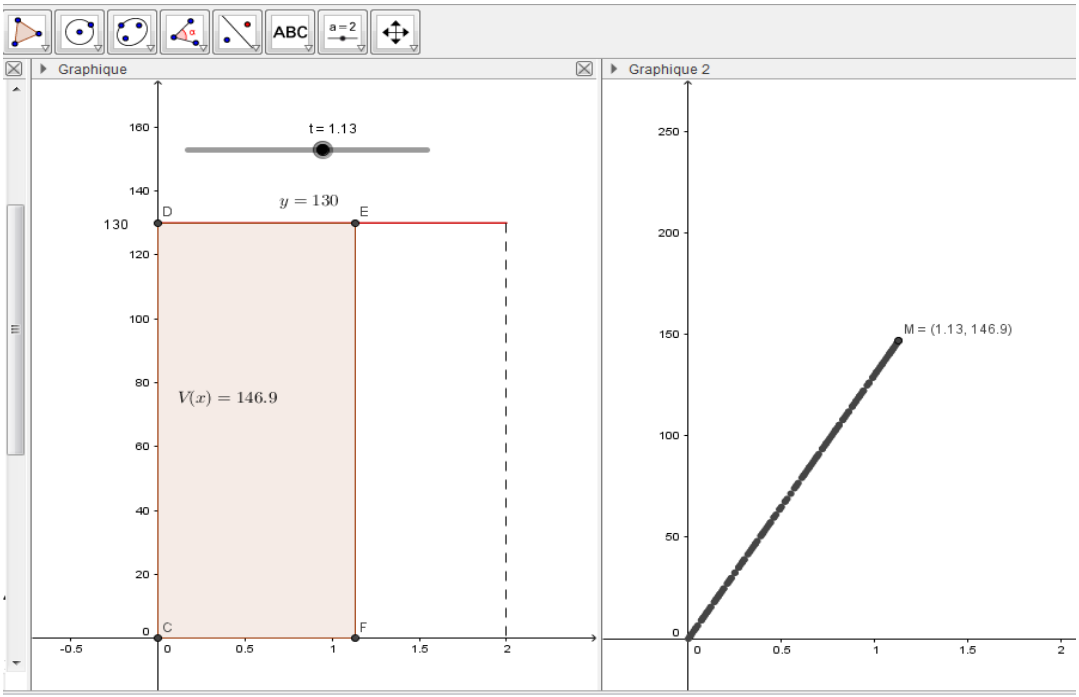
La détermination du volume d'eau s'écoulant revient à calculer l'aire sous la courbe.




### 3ème situation : débit constant au cours du temps, durée variable

On reprend la première situation, mais cette fois on cherche à déterminer à tout instant $t\in[0;2]$, le volume d'eau s'étant écoulé entre les instants 0 et $t$.

D'après ce qui précède, déterminer le volume d'eau écoulé $V(t)$ entre l'instant 0 et l'instant $t$ revient à calculer de l'aire du rectangle $CDEF$ où $CF=t$ et $CD=130$.




On peut tirer avantage de la construction de deux graphiques sous Geogebra ainsi que la combinaison de la fonctionnalité « Afficher la trace » et des curseurs.

Les élèves sont amenés à reconnaître la représentation d'une fonction linéaire, qu'ils peuvent déterminer par lecture graphique. Le calcul de l'aire permet de confirmer la conjecture sur la courbe obtenue.

En effet, on a $V(t) = Aire(CDEF) = t \times 130 = 130t$

On peut alors faire constater le lien entre les fonctions $V$ et $d$. On a clairement $V'(t) = d(t)$

La notion de primitive arrive par le biais du calcul d'aire.

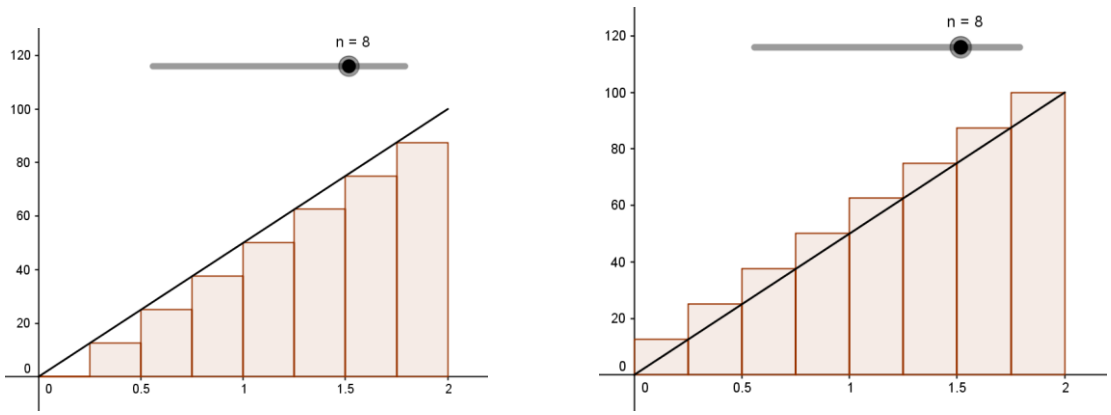
Attention toutefois, le calcul d'aire sous la courbe de la fonction nous donne un volume d'eau dans le contexte.

## 4ème situation : débit fonction linéaire du temps

On suppose que la vitesse dépend du temps de la façon suivante $d(t) = 50t$ en m³/h.

On peut alors amener les élèves à sous-évaluer et surévaluer le volume s'écoulant en utilisant des aires de rectangles. L'utilisation des fonctions « SommeInférieure() » et « SommeSupérieure() » de Geogebra couplées à un curseur permettent d'automatiser ces estimations pour un nombre de rectangles choisi.

Il apparaît que l'amélioration de la précision est corrélée à l'augmentation du nombre de rectangles, ce que l'on peut vérifier grâce à la formule obtenue dans la situation précédente. Il s'ensuit que déterminer le volume exact s'écoulant revient à calculer l'aire sous la courbe de la fonction linéaire $d$ .



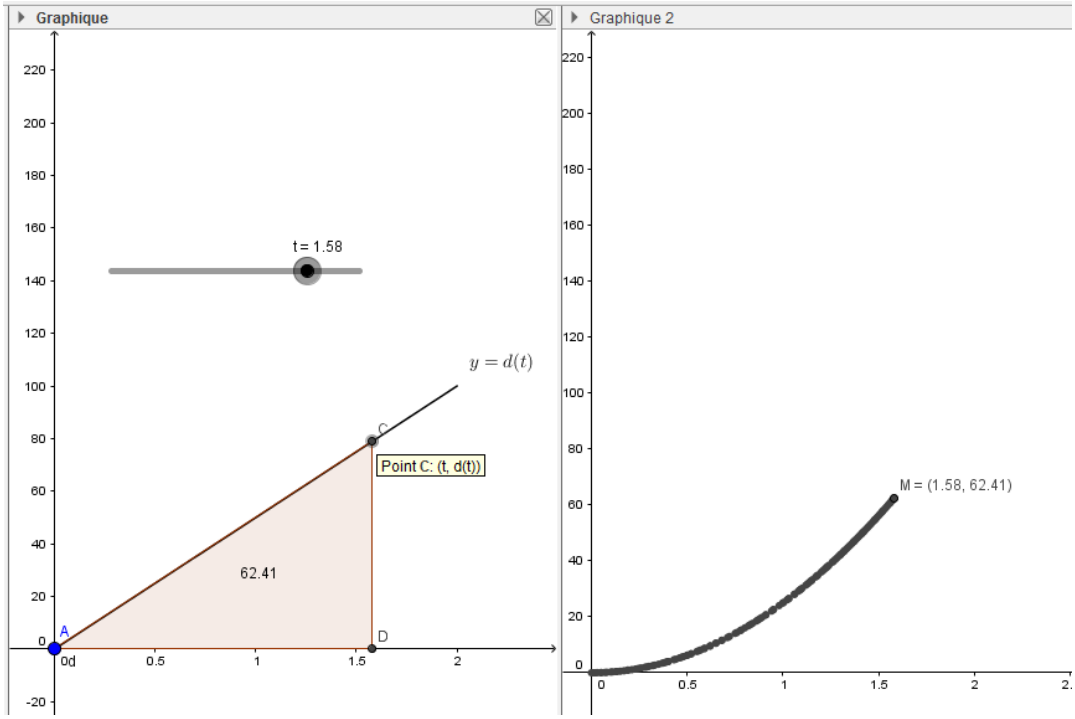
## 5ème situation : débit fonction linéaire du temps, durée variable

La situation est identique à la situation 4, mais on souhaite déterminer le volume d'eau s'étant écoulé entre les instant 0 et $t$, où $t$ appartient à l'intervalle $[0;2]$.

D'après ce qui précède, déterminer le volume d'eau écoulé $V(t)$ entre l'instant 0 et l'instant $t$ revient à calculer de l'aire du triangle $ACD$ où $AD = t$ et $CD = d(t)$.

On peut tirer avantage de la construction de deux graphiques sous Geogebra ainsi que la combinaison de la fonctionnalité « Afficher la trace » et des curseurs.

Les élèves sont amenés à reconnaître une parabole.




Le calcul permet de confirmer la conjecture sur la courbe obtenue. En effet, on a

$V(t) = Aire(ACD) = \frac{1}{2} t \times d(t) = 25t^2$ .

On peut alors faire constater le lien entre l'aire obtenue en fonction de $t$ et la fonction $d$.

En remarquant que $V'(t) = d(t)$, la notion de primitive arrive par le biais du calcul d'aire.

On peut poursuivre avec d'autres dépendances au temps pour lesquelles le calcul exact de l'aire est aisé. Par exemple, avec des fonctions affines ou affines par morceaux. Le calcul exact se limitant à un calcul d'aire de trapèzes.

## 6ème situation : débit fonction du temps

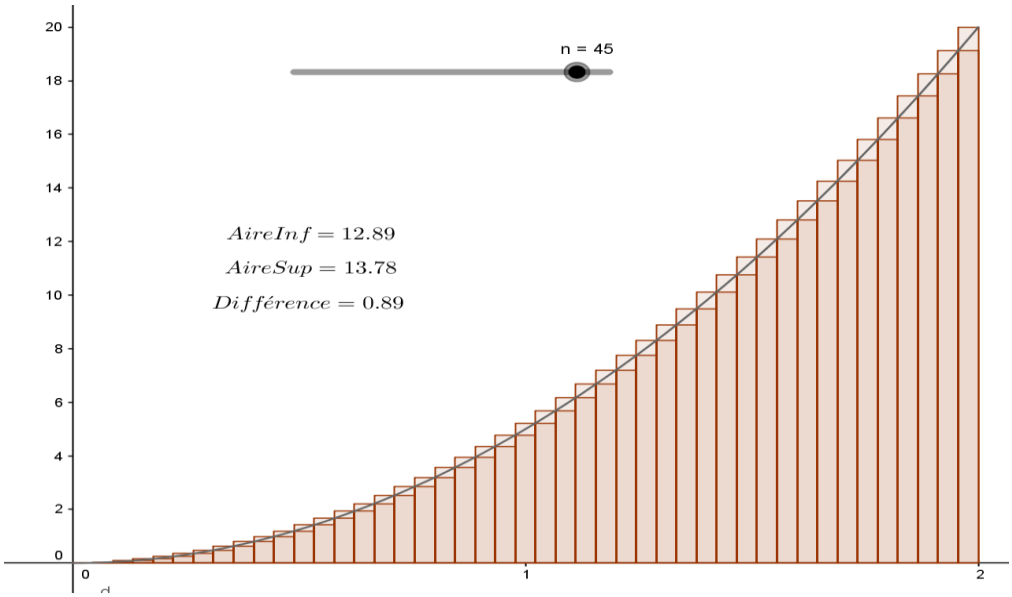
On suppose que les élèves ont acquis l'idée que déterminer le volume d'eau s'écoulant en un temps donné revient à évaluer l'aire sous la courbe du débit. D'autre part, ils ont constaté sur les cas de fonctions linéaires et affines que l'aire sous la courbe s'obtenait par un calcul de primitive.

On suppose que le débit en fonction du temps $t$ est donné par la fonction $d$ définie sur $[0;2]$ par $d(t) = 5t^2$ .

Le choix effectué ici ne sert qu'à illustrer le propos.

On cherche à évaluer dans un premier temps le volume d'eau s'écoulant pendant les deux heures.

Les fonctionnalités « SommeInférieure() » et « SommeSupérieure() » de Geogebra peuvent être utilisées avantageusement pour donner un encadrement de l'aire sous la courbe.




Il apparaît aux élèves que la précision de l'encadrement est corrélé au nombre de rectangles. L'hypothèse d'une valeur exacte de l'aire entre 13,2 et 13,4 peut être émise. Elle s'obtient assez simplement sous Geogebra avec 200 rectangles. L'affichage dynamique des sommes des aires de rectangles permet de faire constater le gain de précision de l'encadrement en fonction du nombre de rectangles.

En adoptant la même stratégie qu'auparavant, on peut alors définir le volume $V(t)$ d'eau s'étant écoulé entre les instant 0 et $t$. On a d'après nos constatations précédentes $V'(t) = d(t)$

Pour déterminer le volume d'eau, il suffit alors de trouver une primitive de $d$ respectant les conditions initiales.

Un travail peut être alors engagé sur la détermination de la constante en fonction de la situation proposée. La valeur exacte, à confronter à l'encadrement, obtenue cette fois uniquement par le calcul de primitive donne une légitimité à développer cet outil.

## Valeur moyenne

Si on veut connaitre la quantité d'eau écoulée, par exemple entre 1h et 2h, cette quantité sera $V(2) - V(1)$.

On a pu établir que $V'(t) = d(t)$. Cette quantité $V(2) - V(1)$ sera notée $\int_1^2 d(t)dt$.

On comprend alors mieux la notion de **valeur moyenne** du débit, entre 1h et 2h par exemple, par l'expression

$$\frac{V(2) - V(1)}{2 - 1} = \frac{\int_1^2 d(t)dt}{2 - 1}$$

## 2.3 Etudier des phénomènes aléatoires

## Le jeu de Monty Hall

Ce jeu, inspiré d'un jeu télévisé et repris de nombreuses fois au cinéma (*Las Vegas 21*) ou à la télévision (*Le Bigdil*), est une situation déclenchante pour permettre à l'élève de mettre en place une démarche de réflexion, s'appuyant sur une simulation avec Python, autour des probabilités conditionnelles.

*Let's make a deal* est un jeu télévisé célèbre aux États-Unis de 1963 à 1986 présenté par Maurice Halprin, dit **Monty Hall**.

Le jeu met en scène 3 portes. Derrière deux d'entre elles se cachent une chèvre et derrière la troisième se cache une voiture, réparties au hasard.

Le candidat va donc tenter de deviner derrière quelle porte se cache la voiture pour tenter de la gagner. Le présentateur Monty Hall, sait ce qui se trouve derrière chacune des portes et le jeu se déroule en plusieurs temps :

- Le joueur choisit tout d'abord une porte au hasard qu'il n'ouvre pas.
- Sur les deux portes restantes, le présentateur en dévoile une derrière laquelle se trouve une des chèvres en sachant quelle est la bonne porte dès le début.
- Puis on demande alors au candidat s'il souhaite conserver son choix ou changer pour sélectionner l'autre porte.

La question est alors la suivante : **le joueur a-t-il intérêt à changer de porte ?**

**Créer un programme Python qui simule une partie de ce jeu**

En collaboration avec le professeur de TIM, cette première partie, peut se réaliser pour partie en classe, mais aussi en dehors de la classe (recherche en temps libre et échange sur les propositions d'algorithmes et programmes).

Ce premier programme peut simplement permettre de réinvestir certaines notions vues en seconde : affectation, test, …

À l'aide de ces programmes et en jouant plusieurs parties successives par groupes d'élèves, il est essentiel de laisser émerger les premières intuitions, travailler sur la formalisation claire (souvent difficile dans les contextes de probabilité) et de débattre notamment sur l'idée reçue sur le fait que : « le choix est équiprobable et que, peu importe son choix, le joueur a 50% de chances de gagner ».

On importe les fonctions « randint » et « seed » du module « random » :

```
from random import randint,seed
seed() #

def gardePorteMH(nbIteration):
    '''Simule une partie du jeu Monty Hall dans le cas où on garde la porte
    et donne la fréquence du nombre de parties gagnées'''
    p=0
    for k in range(nbIteration):
        porte_gagnante = randint(1, 3)
        premier_choix = randint(1, 3)
        if premier_choix == porte_gagnante:# on gagne puisque
                                           # on garde la porte, qui est gagnante
            p=p+1
    return p/nbIteration

def changePorteMH(nbIteration):
    '''Simule une partie du jeu Monty Hall dans le cas où on change la porte
    et donne la fréquence du nombre de parties gagnées'''
    p=0
    for k in range(nbIteration):
        porte_gagnante = randint(1, 3)
        premier_choix = randint(1, 3)
        if premier_choix != porte_gagnante: #on gagne puisque
        # l'on va changer de porte et l'autre porte perdante a été dévoilée
            p=p+1
    return p/nbIteration

print("pour 1000 parties, si on garde la porte, fréquence de gain=",gardePorteMH(1000))
print("pour 1000 parties, si on change la porte, fréquence de gain=",changePorteMH(1000))
```

```
*** Console de processus distant Réinitialisée ***
>>>
pour 1000 parties, si on garde la porte, fréquence de gain= 0.365
pour 1000 parties, si on change la porte, fréquence de gain= 0.696
```

Chaque résultat correspond à une simulation différente, ce qui explique que la somme des fréquences n'est pas égale à 1.

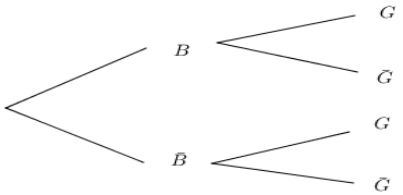
On peut aussi automatiser le lancer de 10 000, 50 000,… parties pour observer la stabilisation des fréquences vers respectivement $\frac{1}{3}$ et $\frac{2}{3}$.

**Formalisation et résolution du problème de Monty Hall :**

Cette partie « exploratoire » soutenue par l'utilisation de Python comme langage textuel a permis à l'élève d'agir, de formuler et de valider. Maintenant la dernière étape permet de démontrer la conjecture émise en utilisant la notion de probabilité conditionnelle et la formule des probabilités totales.

On note B l'événement « la bonne porte est choisie » et G l'événement « la voiture est gagnée »

L'arbre peut ainsi se présenter sous la forme suivante :




La formule des probabilités totales donne : $P(G) = p_B(G)p(B) + p_{\bar{B}}(G)p(\bar{B})$

Dans le cas où l'on garde la porte initialement choisie : $P(G) = \underbrace{p_B(G)}_{1}\ p(B) + \underbrace{p_{\bar{B}}(G)}_{0}\ p(\bar{B}) = \frac{1}{3}$ ($p(B) = \frac{1}{3}$, $p(\bar{B}) = \frac{2}{3}$)

Dans le cas où l'on change la porte initialement choisie : $P(G) = \underbrace{p_B(G)}_{0}\ p(B) + \underbrace{p_{\bar{B}}(G)}_{1}\ p(\bar{B}) = \frac{2}{3}$ ($p(B) = \frac{1}{3}$, $p(\bar{B}) = \frac{2}{3}$)

On a ainsi démontré, ce que notre simulation nous permettait de voir et qui nous semblait contre-intuitif.

**Généralisation et résolution du problème de Monty Hall :**

Des raisonnements qualitatifs ont pu amener certains élèves à trouver la solution dans ce cas. Mais qu'en est-il si, par exemple, on a 5 portes, et que le présentateur décide d'en ouvrir 2 ? Là encore la simulation permet de montrer que la stratégie de changement est la meilleure. Il suffit juste de partir du programme initial et de l'adapter.

```
from random import randint,seed
seed() #

def gardePorteMH(nbIteration):
    '''Simule une partie du jeu Monty Hall dans le cas où on garde la porte
    et donne la fréquence du nombre de parties gagnées'''
    p=0
    for k in range(nbIteration):
        porte_gagnante = randint(1,5)
        premier_choix = randint(1,5)
        if premier_choix == porte_gagnante:# on gagne puisque
                                           # on garde la porte, qui est gagnante
            p=p+1
    return p/nbIteration

def changePorteMH(nbIteration):
    '''Simule une partie du jeu Monty Hall dans le cas où on change la porte
    et donne la fréquence du nombre de parties gagnées'''
    p=0
    for k in range(nbIteration):
        porte_gagnante = randint(1,5)
        premier_choix = randint(1,5)
        if premier_choix == porte_gagnante: #on perd puisque
                                            # l'on va changer de porte
            p=p
        elif randint(0,1)==0: #choix sur les deux restantes
            p=p+1
    return p/nbIteration

print("pour 1000 parties, si on garde la porte, fréquence de gain=",gardePorteMH(1000))
print("pour 1000 parties, si on change la porte, fréquence de gain=",changePorteMH(1000))
```

```
*** Console de processus distant Réinitialisée ***
>>>
pour 1000 parties, si on garde la porte, fréquence de gain= 0.188
pour 1000 parties, si on change la porte, fréquence de gain= 0.405
```

On raisonnera par disjonction de cas.

Dans le cas où l'on garde la porte initialement choisie : $P(G)=\underbrace{p_B(G)}_{1}\,p(B)+\underbrace{p_{\overline{B}}(G)}_{0}\,p(\overline{B})=\frac{1}{5}$

Dans le cas où l'on change la porte initialement choisie : $P(G)=\underbrace{p_B(G)}_{0}\,p(B)+\underbrace{p_{\overline{B}}(G)}_{\frac{1}{2}}\,p(\overline{B})=\frac{2}{5}$

On a ainsi mis en évidence, ce que notre simulation semble toujours indiquer, à savoir que le changement est la meilleure stratégie.

Une différenciation peut être mise en place ici en faisant choisir, pour les élèves qui le désirent, des valeurs de nombre total de portes totales et le nombre de portes que l'on décide d'ouvrir.

Il apparaît que quelles que soient les situations, le changement est la meilleure stratégie.

Pour les élèves les plus habiles avec le formalisme mathématique, on peut même établir les formules générales, pour $n$ portes, $n\geq 3$ avec $p$ portes ouvertes, $1\leq p\leq n-2$.

Dans le cas où l'on garde la porte initialement choisie : $P(G)=\underbrace{p_B(G)}_{1}\,p(B)+\underbrace{p_{\overline{B}}(G)}_{0}\,p(\overline{B})=\frac{1}{n}$

Dans le cas où l'on change la porte initialement choisie : $P(G)=\underbrace{p_B(G)}_{0}\,p(B)+\underbrace{p_{\overline{B}}(G)}_{\frac{1}{n-(p+1)}}\,p(\overline{B})=\frac{1}{n-(p+1)}\frac{n-1}{n}$

Cette dernière probabilité est toujours supérieure à la précédente, d'autant plus que l'on ouvre plus de portes.

## Un exemple de simulation pour comprendre la fluctuation d'échantillonnage

Lucien March écrivait, en 1930, « Il est toujours dangereux d'appliquer des formules dont on n'a bien saisi ni le sens profond, ni l'esprit ».

Pour comprendre la problématique de la prise de décision, il est primordial que les élèves prennent conscience des fluctuations d'échantillonnage. Puis, qu'ils comprennent comment fluctuent les fréquences d'un caractère. On s'arrête à la notion de fréquences dans le secondaire.

Avant de « simuler $N$ échantillons de taille $n$ d'une loi binomiale et représenter les fréquences observées des 1 par un histogramme ou un nuage de points », on pourra faire manipuler ces fluctuations aux élèves.

On peut facilement fabriquer un outil de simulation, comme le suggère l'article « Un biberon comme outil de simulation au lycée » du bulletin vert de l'APMEP n°500 (Les auteurs tiennent cette idée d'un collègue de l'enseignement agricole ! Il se reconnaîtra…).
Cet outil, une bouteille de lait opaque, dont le bouchon est découpé pour introduire une tétine transparente, contient des boules de cotillons de couleur. La proportion $p$ de boules rouges est modifiable à volonté.

**Activité 1 « A la main »**

On remplit les bouteilles avec une certaine proportion $p$ de boules rouges et de boules d'une autre couleur.

Dans un premier temps, la proportion de boules rouges est connue. Dans le cas étudié, $p = \frac{2}{5}$.

Le nombre de boules rouges obtenu dans un échantillon de taille $n$ prélevé dans la population est une variable aléatoire $X_n$. L'objectif est de comprendre et d'illustrer la variabilité de la fréquence $F_n = \frac{X_n}{n}$ d'apparition des boules rouges dans l'échantillon.

Un échantillon de taille $n$ est constitué des résultats de $n$ répétitions indépendantes de la même expérience.

**Chaque élève génère 4 échantillons avec remise de 20 boules dans la bouteille.**

Pour chaque échantillon, il note les résultats (1 si la boule est rouge, 0 sinon), puis calcule la valeur de $F_n$.

Cette étape est un peu longue mais elle permettra de disposer d'un certain nombre d'échantillons et la manipulation est essentielle pour comprendre ce qui est fait avant une simulation informatique.

| $F_n$ | Boule **éch n°** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0,45 | **1** | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0,55 | **2** | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0,5 | **3** | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0,6 | **4** | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

Pour une classe de 25 élèves, on disposera de 100 échantillons et donc de 100 valeurs de $F_{20}$.

Après une mise en commun des résultats des élèves de la classe, on pourra :

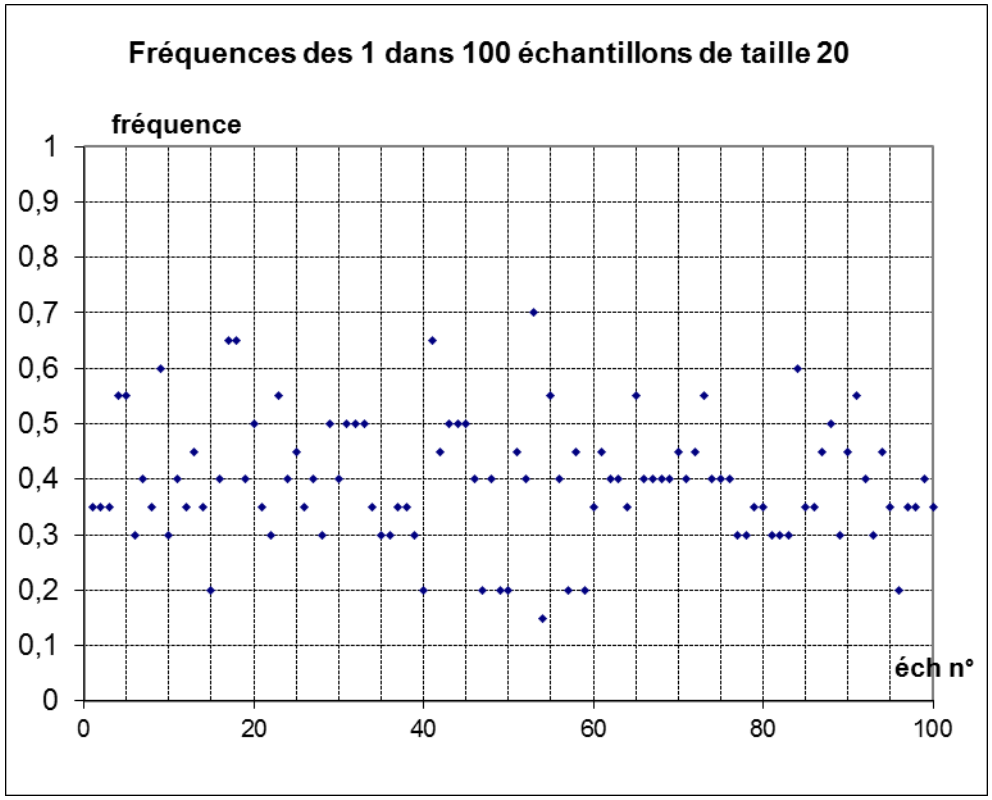
- calculer l'écart-type *s* des fréquences observées,
- dessiner le nuage de points des fréquences observées,

- calculer le pourcentage d'échantillons pour lesquels $F_n \in [p-s; p+s]$, $F_n \in [p-2s; p+2s]$ et $F_n \in [p-3s; p+3s]$ .

On remarque notamment que $P(p-2s \leq F_n \leq p+2s) \approx 0,95$ ou encore que pour environ 95% des échantillons, la fréquence $F_n$ appartient à l'intervalle $\left[p-\frac{1}{\sqrt{n}}; p+\frac{1}{\sqrt{n}}\right]$.

➢ On pourra visualiser ces pourcentages sur le nuage de points en traçant les droites d'équation $y = p \pm s$, $y = p \pm 2s$ et $y = p \pm 3s$

Pour le nuage de points donné, $p = 0,4$ et $s \approx 0,11$




Cette activité permettra aux élèves de bien comprendre la fluctuation d'échantillonnage et sera l'occasion de faire percevoir la diversité des interprétations possibles selon l'échantillon prélevé.

Par exemple : pour l'échantillon n°53, la fréquence observée des 1 est loin de 0,4 et on pourrait douter pour cet échantillon que la bouteille contienne $\frac{2}{5}$ de billes rouges, tandis que pour l'échantillon n°99, la fréquence observée des 1 est de 0,4, donc on ne douterait absolument pas du fait que $p = \frac{2}{5}$.

$[p-2s; p+2s] = [0,18; 0,62]$. Au vu de cette simulation, 5 échantillons sur 100 donnent des valeurs de $F_n$ en dehors de l'intervalle $[p-2s; p+2s]$.

On retrouve un résultat théorique qui indique que, pour *n* assez grand, c'est pour 5% des échantillons que la valeur observée $F_n \notin [p-2s; p+2s]$ ou que pour 95% des échantillons, la valeur observée $F_n \in [p-2s; p+2s]$

L'activité suivante sur les grains de blé développera comment réaliser des simulations analogues à ce qui est fait avec un tableur pour beaucoup plus d'échantillons.

Toutefois, on peut déjà avoir une représentation graphique de telles simulations de ce type d'expérience :

Grâce au package numpy, les commandes

```
liste_grains=[]
for i in range(n):
    liste_grains.append(random.choice([0,1],p=[0.6,0.4]))
```

permettent de créer une liste de 0 et de 1 avec une probabilité définie (respectivement 0,6 et 0,4).

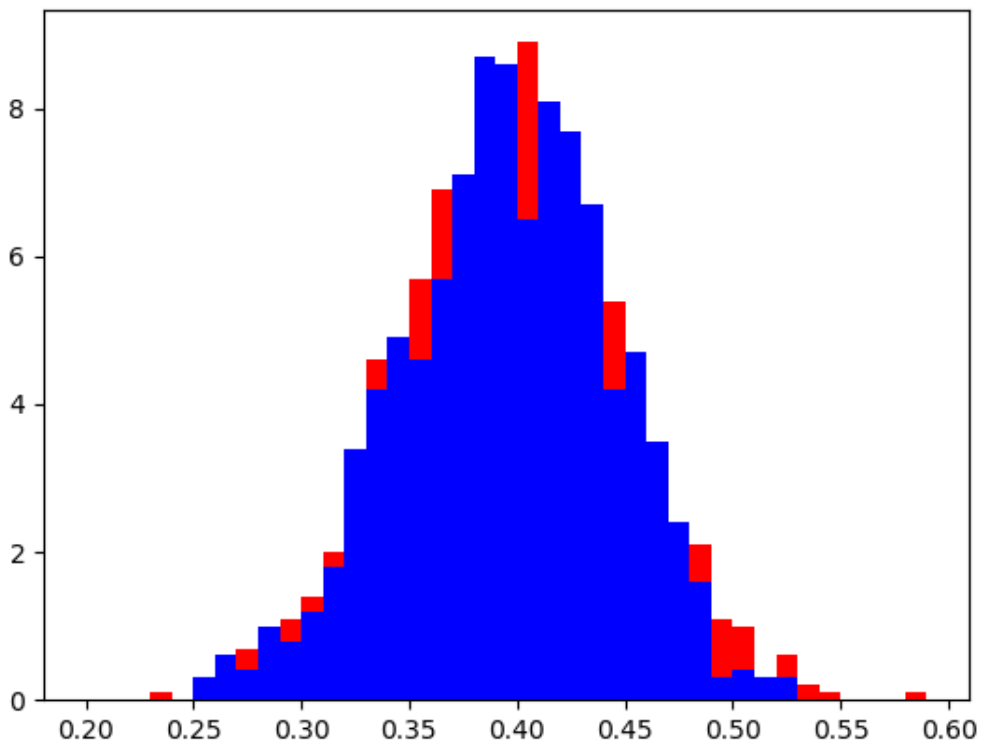
On peut alors déterminer la fréquence d'apparition des 1 de cette liste et répéter cela un certain nombre de fois.

On peut alors définir une fonction qui génère *m* échantillons **liste_grains[ ]** de taille *n* de 0 et de 1 qui apparaissent suivant une probabilité définie et qui retourne la liste des moyennes dans la liste **moyenneslistes[ ]**.

```
from numpy import*
from matplotlib.pyplot import*

def statechant(nbechant,taillechant):
    moyenneslistes=[]
    for j in range(nbechant):
        liste_grains=[]
        for i in range(taillechant):
            liste_grains.append(random.choice([0,1],p=[0.6,0.4]))
        moyenneslistes.append(mean(liste_grains))  # mean pour moyenne
    return moyenneslistes
```

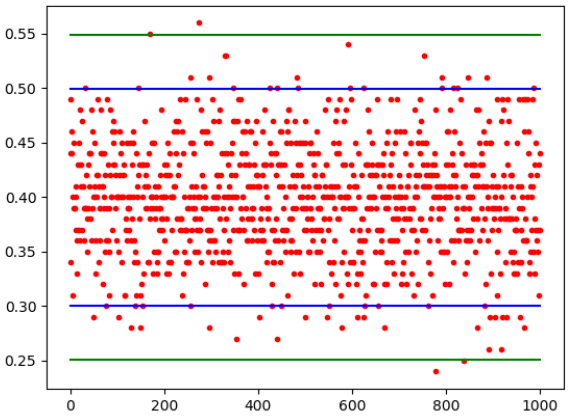
Il ne reste qu'à faire afficher l'histogramme des fréquences grâce à l'option normed=True (deux histogrammes sont superposés pour montrer qu'il y a peu de variations).

```
#1000 échantillons de taille 100
hist(statechant(1000,100),bins=arange(0.2,0.6,0.01),color='red',normed=True)
hist(statechant(1000,100),bins=arange(0.2,0.6,0.01),color='blue',normed=True)
show()
```



**std** donne l'écart-type de la série

```
#1000 échantillons de taille 100
e=statechant(1000,100)
plot(e,"r.")
plot([0,1000],[0.4-2*std(e),0.4-2*std(e)],"b")
plot([0,1000],[0.4+2*std(e),0.4+2*std(e)],"b")
plot([0,1000],[0.4-3*std(e),0.4-3*std(e)],"g")
plot([0,1000],[0.4+3*std(e),0.4+3*std(e)],"g")
show()
```

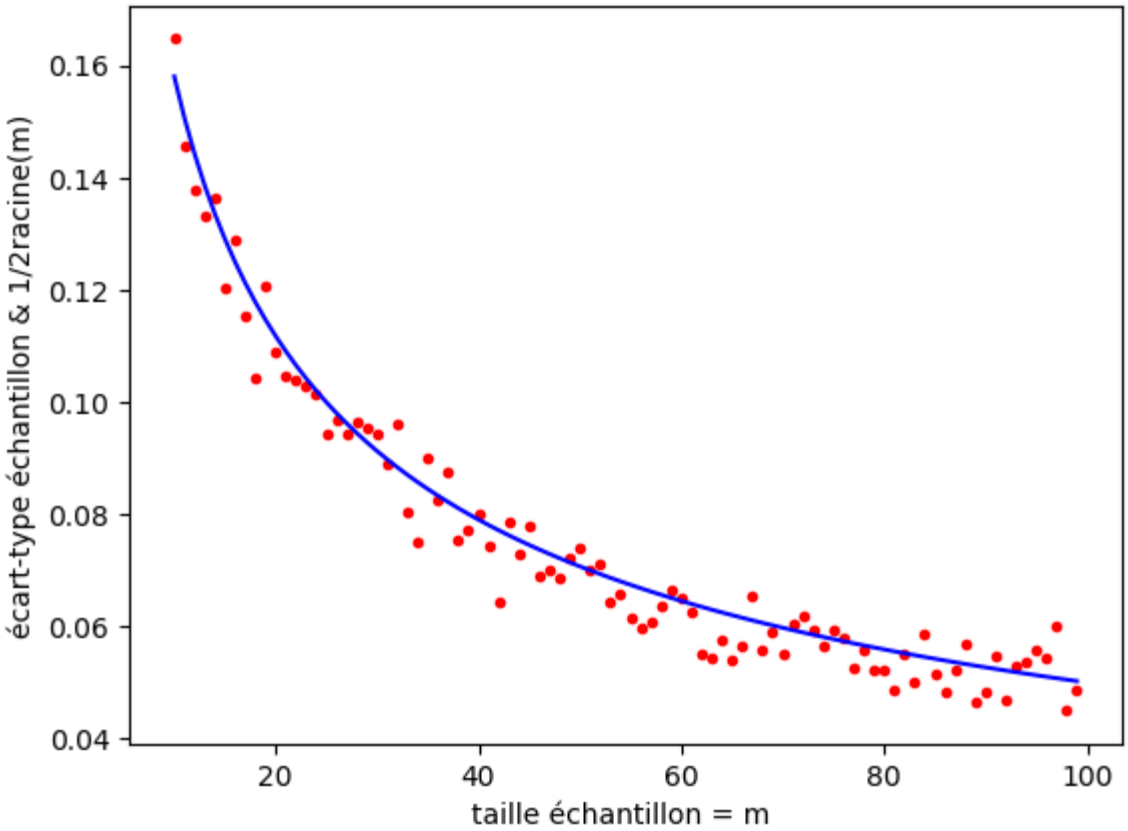



On pourra faire observer que $2s \approx \frac{1}{\sqrt{n}}$

On utilisera les bibliothèques numpy et matplotlib.pyplot pour avoir les fonctions de statistiques et de représentations graphiques.

```
from numpy import*
from matplotlib.pyplot import*

def liste_ecartypechant(m,n):
    moyenneslistes=[]
    for j in range(m):
        liste_grains=[]
        for i in range(n):
            liste_grains.append(random.choice([0,1],p=[0.6,0.4]))
        moyenneslistes.append(mean(liste_grains))
    return std(moyenneslistes)  # std est l'écart type

ecartypechant=[]
taillechant=range(10,100)
for j in range(10,100):
    ecartypechant.append(liste_ecartypechant(100,j))
plot(taillechant,ecartypechant,"r.")  #"r." fait des points rouges
z=[1/(2*sqrt(x)) for x in taillechant]
plot(taillechant,z,"b") #"b" pour blue
xlabel("taille échantillon = m ")
ylabel("écart-type échantillon & 1/2racine(m)")
show()
```

On fait alors afficher les valeurs de chaque écart-type calculé pour un échantillon de taille $m$ que l'on compare à la courbe théorique des $m \mapsto \frac{1}{2\sqrt{m}}$ superposée aux écarts-types de chaque échantillon.




Ainsi, plus l'échantillon est grand, plus sûre est la valeur estimée puisque l'écart-type diminue.

## Un exemple pour comprendre la loi binomiale.

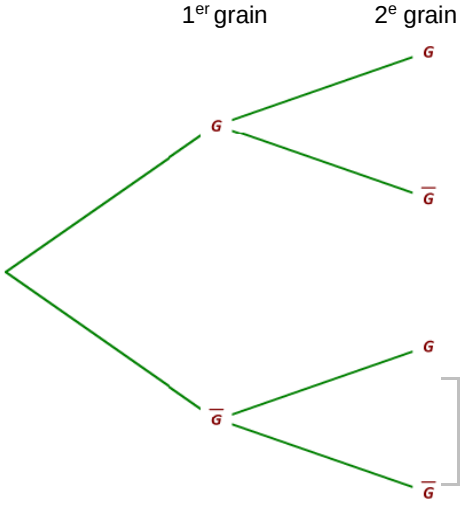
En présence d'un vieux stock de grains de blé à semer, on cherche à étudier le pouvoir germinatif des grains qu'il contient, c'est-à-dire le pourcentage des grains qui vont germer lorsqu'ils seront plantés.

Pour $x$ grains de blé posés sur du papier filtre dans une boîte de Petri, on mouille du papier filtre laissé à la lumière et à bonne température (24 °C). Au bout de 2 jours, le grain devrait germer. Si au bout de 5 jours, le grain ne germe pas, on considère qu'il ne germera jamais. S'il germe mal (pas de racine ou pas de feuille), on considère qu'il ne germe pas.

Le problème est de savoir comment conclure à partir du résultat de cette expérience. En effet, il y a une part de hasard lors de la prise des grains et il y aura une fluctuation des résultats selon l'échantillon choisi.

Établissons une méthodologie :

- regardons, tout d'abord, comment faire à partir d'un stock connu dont le pouvoir germinatif est de 50 %.
- étudions ce qui se passe à partir d'un stock connu dont le pouvoir germinatif n'est pas de 50 %, par exemple 75 %.
- étudions ce qui arrive avec un échantillon de grains suffisamment grand.




**Première approche: Le pouvoir germinatif est connu et vaut 50 %**

Soit un échantillon de 2 grains pour commencer.

Soit $G$ l'événement : « le grain germe » et $\overline{G}$ l'événement « le grain ne germe pas ».

A partir de l'arbre des possibles ci-contre, on peut compléter le tableau ci-dessous :

| Nombre de grains qui germent | 0 | 1 | 2 |
|---|---|---|---|
| fréquence | 0,25 | 0,5 | 0,25 |

et déterminer pour notre échantillon la probabilité (ici égale à la fréquence) que celui-ci nous donne le bon pouvoir germinatif.

On peut faire de même avec un échantillon de 3 grains :

| Nombre de grains qui germent | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| fréquence | 0,125 | 0,375 | 0,375 | 0,125 |

Il est difficile d'obtenir le pouvoir germinatif du sac avec seulement 3 grains.

Il est donc indispensable de prendre davantage de grains dans notre échantillon, mais la contrepartie est qu'il est bien long de le faire à la main.

On peut alors résoudre cette situation en simulant la germination des grains à l'aide de Python.

**Une première simulation avec un pouvoir germinatif de 50%.**

On utilisera les bibliothèques math, numpy et matplotlib.pyplot pour avoir les fonctions de statistiques et de représentations graphiques.

```
from math import*
from numpy import*
from matplotlib.pyplot import*
```

La fonction random.choice permet de choisir au hasard un élément de la liste donnée avec équiprobabilité. On aurait pu utiliser d'autres fonctions mais celle-ci a d'autres qualités que nous allons découvrir.

La commande suivante permet de simuler si un grain germe ou non, le choix se faisant de manière aléatoire.

```
germe = random.choice(["non","oui"])
print(germe)
```

Pour simuler la germination de 10 grains, on peut saisir :

```
germe = random.choice([0,1],10)
print(germe)
```

Le faire plusieurs fois permet de constater le côté aléatoire.

Si on cherche à compter le nombre de « oui » dans chaque échantillon (on peut écrire un programme mais on souhaite aller plus vite), il suffit de changer le codage de l'état du grain.

Comme l'intérêt est d'avoir des grains qui germent, le grain qui germe est codé avec le nombre 1 et le grain qui ne germe pas avec le nombre 0. La raison d'un tel codage est simple, pour compter le nombre de grains qui germent, il suffit d'ajouter tous ces nombres.

```
germe = random.choice([0,1],10)
print(germe)
print(sum(germe))
```

La simulation de la prise d'un échantillon de 3 grains permet de tester le pouvoir germinatif.

En utilisant la commande « str(random.choice([0,1],3)) », on simule cette expérience et on l'écrit dans une chaîne de caractères.

Le programme suivant permet de faire 1000 simulations et de toutes les enregistrer dans une liste nommée ***experiences*** et d'afficher la liste en fin de programme.

```
experiences = []
for compteur in range(1000):
    experiences.append(str(random.choice([0,1],3)))
print(experiences)
```

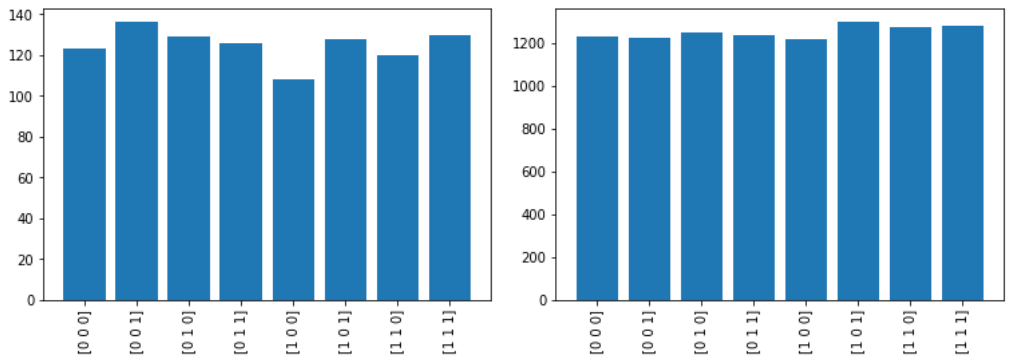
On observe alors tous les chemins que peuvent prendre les expériences en suivant les branches de l'arbre des possibles que nous avons fait lors de la première approche.

On cherche alors les branches plus représentées que d'autres.

Pour répondre à cette question, nous allons représenter la répartition des différentes possibilités à l'aide de la fonction suivante nommée **graphique** (car Python n'a pas par défaut de module de représentation)

```
def graphique(Donnees):
    Valeurs=list(sorted(set(experiences))) # set enlève les doublons, sorted classe
    Effectifs=[experiences.count(i) for i in Valeurs]
    bar(Valeurs, Effectifs)
    xticks(rotation=90)
    show()

graphique(experiences)
```

Il y a une fluctuation dans les répartitions sur deux simulations mais les différentes possibilités sont sensiblement équivalentes. On peut le constater en augmentant le nombre de simulations.

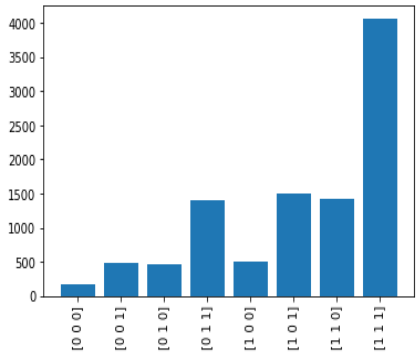



**Une deuxième simulation avec un pouvoir germinatif de 75%.**

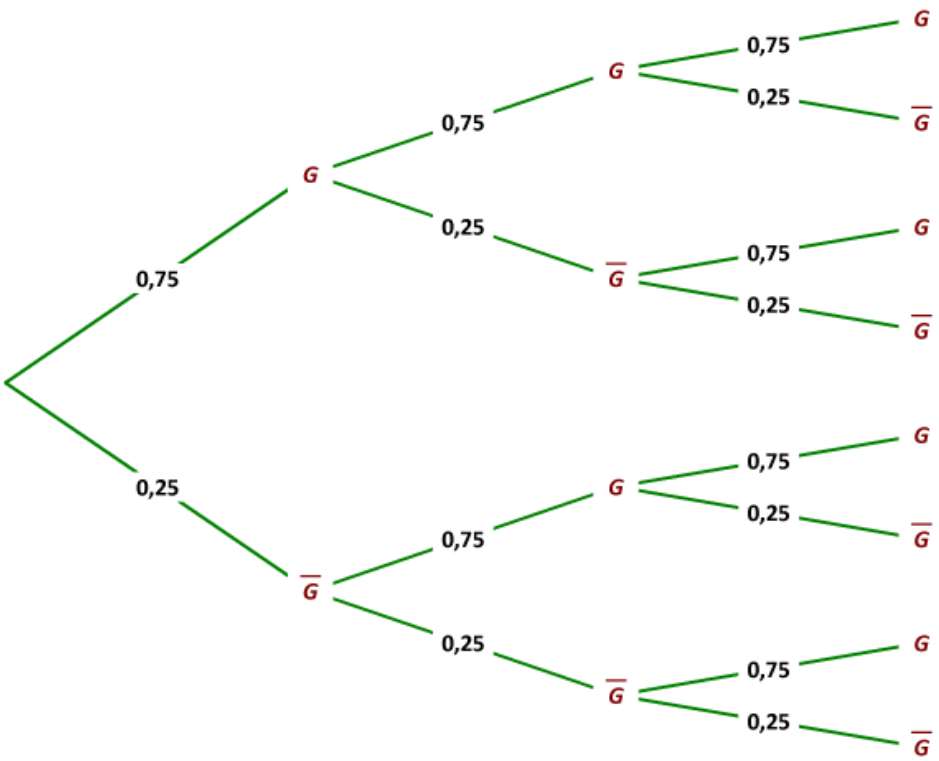
Simulons alors l'expérience en imposant les probabilités pour les deux cas avec l'option « p=[0.25,0.75] » dans la fonction « random.choice ».

Si on reprend le même programme que le précédent, légèrement modifié pour la circonstance :

```
experiences = []
for compteur in range(10000):
    experiences.append(str(random.choice([0,1],3,p=[0.25,0.75])))
graphique(experiences)
```



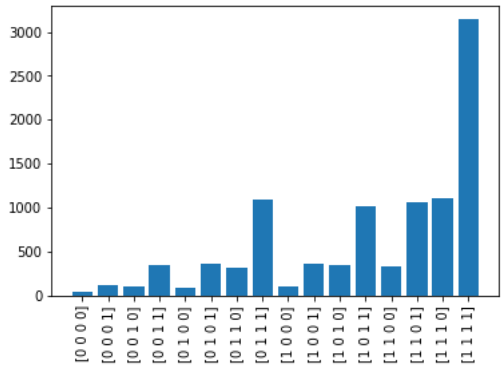

En s'appuyant sur l'arbre pondéré, on retrouve le cas [1,1,1] qui est le plus fréquent et cela s'explique à l'aide des fréquences ou probabilités sur les branches de l'arbre ci-dessous en utilisant les connaissances des élèves sur les proportions de proportions (voir « ***automatismes » du programme de la filière technologique E.N.***). Par exemple, pour la deuxième branche, $G-G-\overline{G}$, en la lisant de droite à gauche, 25 % des échantillons qui ont vu leurs deux premiers grains germer n'ont pas vu leur dernier grain germer. C'est-à-dire que 25 % des 75 % des échantillons qui ont vu leur premier grain germer ont vu le second germer mais pas le troisième. Finalement 25 % des 75 % des 75 % des échantillons ont vu leurs deux premiers grains germer mais pas le troisième.




Ces calculs expliquent des cas de même fréquence, comme [0,1,1], [1,0,1], [1,1,0].

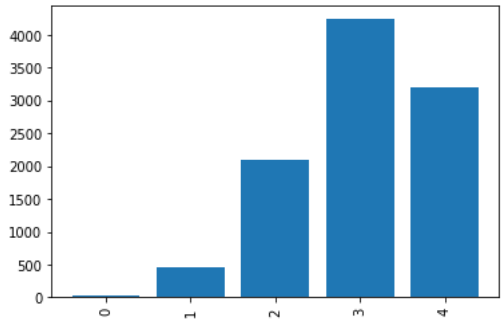
On peut continuer avec l'arbre pondéré pour une expérience avec 4 grains de blé. (On pourra compléter l'arbre donné précédemment) et vérifier les conjectures en modifiant le programme pour faire des simulations avec 4 grains.

```
experiences = []
for compteur in range(10000):
    experiences.append(str(random.choice([0,1],4,p=[0.25,0.75])))
graphique(experiences)
```




Maintenant, au lieu de regarder les branches obtenues par la simulation, regardons le nombre de grains qui germent (à l'aide de la fonction ***sum*** dans chaque expérience et conservons le tout pour le représenter.

```
experiences = []
for compteur in range(10000):
    experiences.append(sum(random.choice([0,1],4,p=[0.25,0.75])))
graphique(experiences)
```



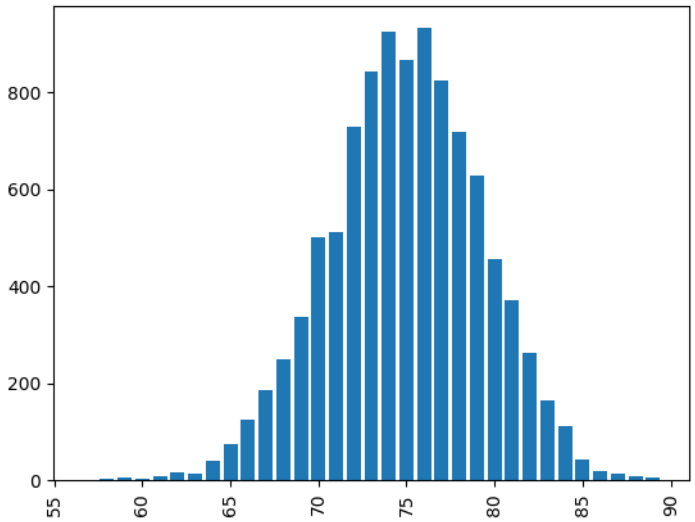

On peut alors faire réfléchir au moyen de prédire la hauteur des bâtons à partir de l'arbre.

**Une troisième simulation avec de grands échantillons et un pouvoir germinatif de 75%.**

Testons maintenant 100 grains pour connaître plus facilement le pouvoir germinatif.

Simulons 10000 expériences et affichons la répartition des différents résultats pour des tests de 100 grains avec un pouvoir germinatif de 75%.

```
experiences = []
for compteur in range(10000):
    experiences.append(sum(random.choice([0,1],100,p=[0.25,0.75])))
graphique(experiences)
```



D'après le graphique obtenu, on peut faire réfléchir les élèves sur différentes questions :

- est-il possible qu'aucun grain ne germe ?
- comment calculer la fréquence de cet événement à l'aide de la méthode trouvée précédemment ?
- est-il possible de n'avoir que 65 grains qui germent alors que le pouvoir germinatif est de 75 % ?
- est-il possible d'avoir 85 grains qui germent alors que le pouvoir germinatif est de 75 % ?

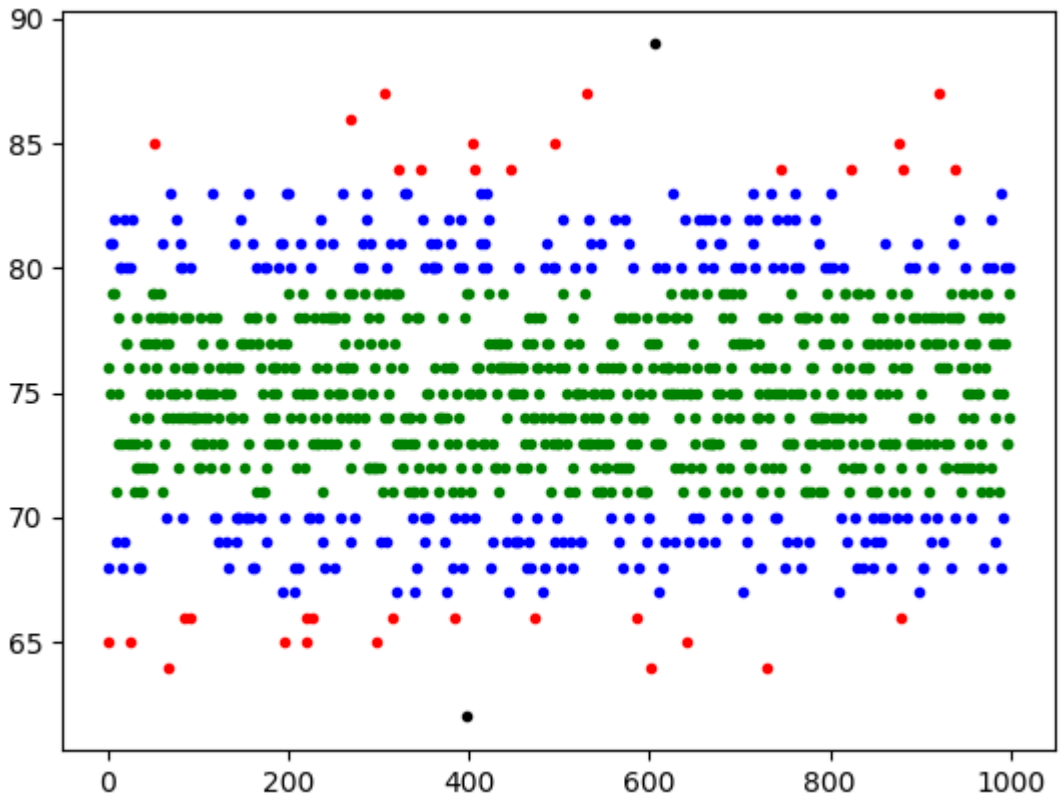
On peut faire constater dans ce cas que la fréquence de germination dans l'échantillon est de 75%, généralement plus ou moins 10%, le terme « généralement » restant à expliciter collectivement.

**Notion d'intervalle de fluctuation.**

Soit s l'écart type des nombres de grains germés dans les échantillons de l'expérience.

On peut améliorer le programme suivant pour colorier en vert, les nombres de grains germés compris dans l'intervalle [75- s ; 75+s], en bleu ceux compris dans [75- 2s ; 75+2s] mais pas dans [75- s ; 75+s], en rouge ceux compris dans [75- 3s ; 75+3s] mais pas dans [75- 2s ; 75+2s] et en noir les autres.

```
ectype = sqrt(mean(abs(experiences - mean(experiences))**2))

for compteur in range(len(experiences)):
    if abs(experiences[compteur]-75)<ectype:
        plot(compteur,experiences[compteur],'g.')
    elif abs(experiences[compteur]-75)<2*ectype:
        plot(compteur,experiences[compteur],'b.')
    elif abs(experiences[compteur]-75)<3*ectype:
        plot(compteur,experiences[compteur],'r.')
    else:
        plot(compteur,experiences[compteur],'k.')
show()
```




Est-il raisonnable de dire, qu'en général, les nombres de grains germés sont dans l'intervalle [75- 2s ; 75+2s] ?

Pour cela, on modifie le programme pour compter le nombre de points verts ou bleus.

```
compteurVertBleu = 0
for compteur in range(len(experiences)):
    if abs(experiences[compteur]-75)<2*ectype:
        compteurVertBleu = compteurVertBleu + 1
print( compteurVertBleu / len(experiences) )
```

La part des points verts ou bleus tourne autour de 95 %. Nous avons donc trouvé avec [75- 2s ; 75+2s] un intervalle de fluctuation dans lequel on trouve le nombre de grains germés dans environ 95 % des échantillons.

On peut comparer ce dernier résultat à celui du programme suivant qui ramène aux constats de la troisième simulation :

```
compteurIntervalle = 0
for compteur in range(len(experiences)):
    if abs(experiences[compteur]/100-0.75)<0.1:
        compteurIntervalle = compteurIntervalle + 1
print( compteurIntervalle / len(experiences) )
```

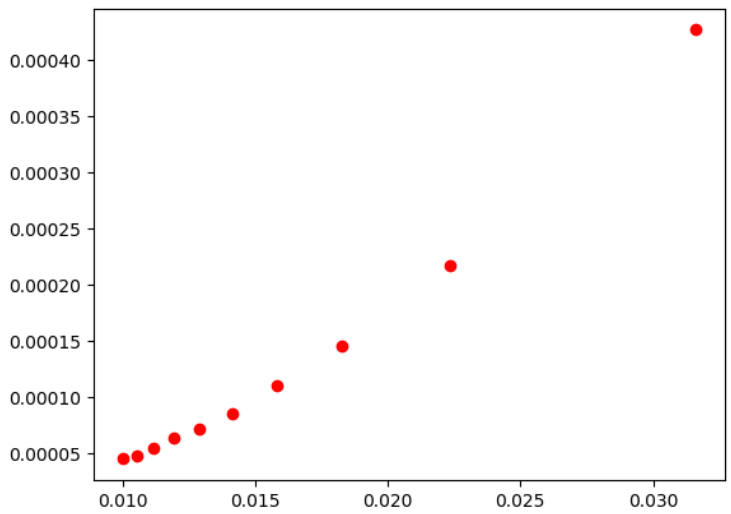
**Vers une formule simplifiée de l'intervalle de fluctuation**

Dans la situation précédente, nous avons trouvé, en généralisant, que si $p$ est le pouvoir germinatif de nos grains et s l'écart type des échantillons, alors $[p-2s; p+2s]$ est l'intervalle dans lequel vont fluctuer les fréquences des grains germés dans 95% des échantillons.

Mais pour connaître s, il faut prendre beaucoup d'échantillons. Un autre candidat pour être un intervalle de fluctuation semble être $\left[p-\frac{1}{\sqrt{n}}; p+\frac{1}{\sqrt{n}}\right]$ dans lequel semble fluctuer les fréquences des grains germés dans plus de 95% des échantillons, comme cela a été vu dans l'activité précédente.

Essayons de vérifier le lien d'une autre façon en représentant pour différentes tailles d'échantillons l'écart type des résultats de celui-ci.

```
for n in [1000,2000,3000,4000,5000,6000,7000,8000,9000,10000]:
    experiences = []
    for compteur in range(1000):
        experiences.append(sum(random.choice([0,1],p=[0.25,0.75]))/n)
    plot(1/sqrt(n),std(experiences),'ro') #point rouge
show()
```




Les points semblent alignés et en estimant cette équation de cette droite, on obtient que $\frac{1}{\sqrt{n}}$ est environ égal à 2s

Ainsi, pour diviser la dispersion par k, on doit multiplier la taille de l'échantillon par k².

Cela permet d'avoir une idée du nombre de grains de blé à prendre dans notre échantillon pour estimer le pouvoir germinatif du blé du sac à 1% près.