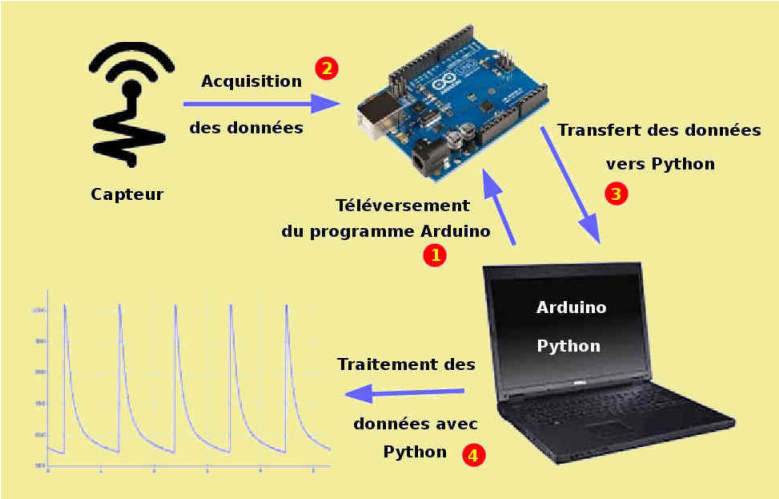
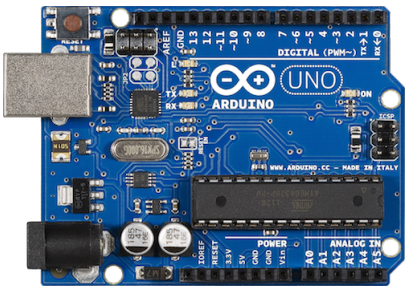


Formation distancielle

La physique COMPUTATIONNELLE au lycée



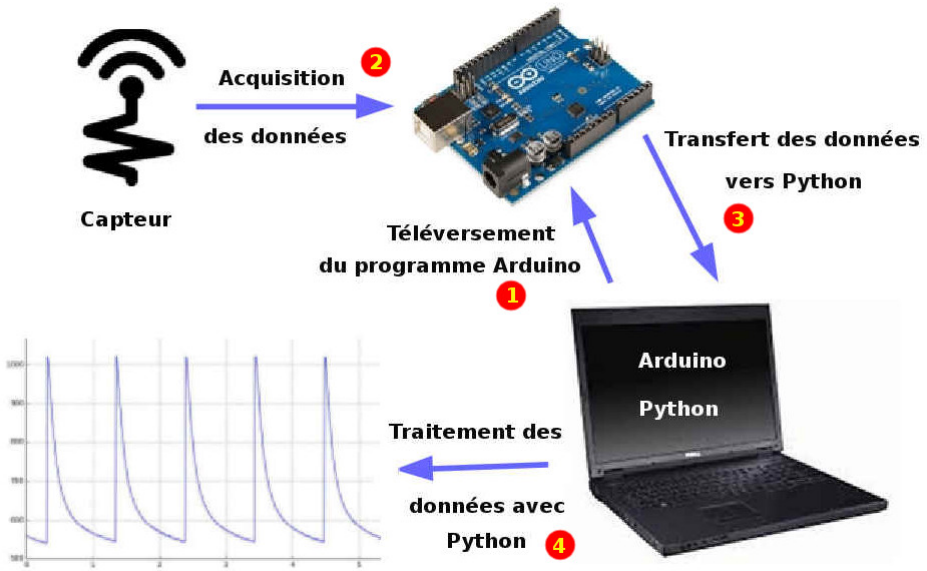
Christophe Casseau
2017-2018

IA-IPR : David Boyer
Équipe de formation : Laurent Sartre
Denis Dumora
Marc Edlin

Table des matières

Introduction

L'objectif de cette formation est d'apprendre à échanger des informations entre la carte **Arduino UNO** et votre ordinateur à travers le langage de programmation **Python**. Pour un coût extrèmement modique, environ 20 euros pour une carte Arduino, on dispose d'un système d'acquisition performant, évolutif et paramétrable en fonction des besoins de l'utilisateur.



Ce document comporte une première partie consacrée à l'installation et à la découverte de la carte *Arduino UNO*. Vous apprendrez quelques bases essentielles de la programmation Arduino en vue de piloter les capteurs connectés à la carte.

La deuxième partie permet de s'initier à la communication entre la carte *Arduino UNO* et le langage de programmation Python afin de récupérer les données collectées par les capteurs.

La dernière partie permet de mettre en oeuvre des exemples d'acquisition de données à partir d'un capteur (température, lumière, ultrason, ...) connecté à la carte *Arduino UNO* et d'effectuer un traitement numérique de ces données avec Python. Les activités proposées s'inspirent des programmes du lycée.

En ANNEXE je propose quelques solutions (totales ou partielles) aux exercices et activités que vous rencontrerez tout au long de votre lecture.

Comment lire ce document ?

Tous les participants à cette formation reçoivent un kit contenant :

- une carte Arduino UNO
- un mini-prototype circuit breadboard 400 points
- une diode RVB
- un bouton poussoir
- une photorésistance
- une thermistance
- un capteur ultrason HC-SR04
- 3 résistances 330 Ω
- 1 résistance 10 $k\Omega$
- 6 fils de connexion

Le kit devra être rendu le jour de la formation à Bordeaux au lycée Camille Jullian. Il va vous permettre de tester tranquillement chez vous toutes les activités expérimentales proposées dans ce document. En cas de problème insoluble, vous avez également la possibilité de poser une question sur un espace privé de gestion de projet : Trello, nous ferons notre possible pour répondre rapidement.
Tout au long de ce document vous rencontrerez également des liens web de couleur rose. Ils permettent d'avoir un complément d'informations et/ou d'approfondir certaines connaissances.

Remerciements

Je remercie Michel Fourtinon professeur au lycée Camille Jullian de Bordeaux et Hervé Perrin professeur en C.P.G.E au lycée Camille Jullian de Bordeaux d'avoir pris le temps de relire et de tester l'ensemble des activités proposées. Leurs idées et leurs suggestions m'ont permis d'ajouter certains éléments importants de ce document.

Je remercie également Jean-Luc Charles enseignant chercheur à l'ENSAM de Bordeaux, pour les documents qu'il m'a fournis ainsi que le temps qu'il a passé à répondre à toutes mes questions de la manière la plus rigoureuse possible avec la toute la compétence et la gentillesse qui le caractérise.

Christophe Casseau

1 La pensée informatique en sciences physiques

1.1 La révolution numérique

La pensée informatique a déjà influencé l'approche de la recherche en sciences. Cela c'est traduit par son intégration dans les grands domaines d'application de la physique et de la chimie, ainsi que dans leur enseignement avec la création de cours de physique ou chimie computationnelle offrant de nouvelles méthodes de calcul, de nouveaux outils et techniques pour enrichir et comprendre la physique du XXI^{e} siècle. Le *machine learning* (que l'on pourrait traduire en français par : apprentissage automatique de la machine) associé à l'émergence et au développement des technologies Big Data ouvre de nouvelles perspectives dans les domaines de la physique et de la chimie tel que la résolution de problèmes de physique quantique en matière condensée, de problèmes d'astrophysique ou encore dans la simulation numérique de systèmes physiques complexes comme la création de jumeaux numériques de réacteurs nucléaires. De plus l'informatique à travers des concepts comme l'algorithmique, la science des données, l'architecture des ordinateurs ou encore la programmation est omniprésente en sciences mais également dans d'autres disciplines comme la linguistique, l'analyse de l'apprentissage ou l'étude du comportement humain dans leurs efforts de modélisation. L'ensemble de ces méthodes informatiques transposables aux autres disciplines afin de répondre à des questions qui leur sont propres est appelé *Computationnal Thinking* ou encore *Pensée Informatique ou Computationnelle*.

1.2 La pensée informatique ou pensée computationnelle un savoir fondamental ?

La pensée informatique ne possède pas à notre connaissance de définition précise mais semble regrouper un ensemble de compétences et de savoirs faire qui ne sont pas réservés aux seuls informaticiens. Jeanette Wing définit la pensée informatique de la façon suivante :

Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine.

La vidéo : What is Computationnal Thinking by Jeanette Wing

À ce titre, nous attirons l'attention du lecteur sur la nécessité de ne pas confondre Pensée Informatique et programmation. En effet, la programmation utilise un langage pour implémenter une solution induite par la Pensée Informatique. Aujourd'hui la pensée informatique a investi la plupart des domaines de recherche et d'application Elle fait dorénavant partie des cursus d'Ecoles d'Ingénieurs ou d'Université prestigieuses. L'EPFL (École Polytechnique Fédérale de Lausanne) propose depuis 2013 à la liste de ses enseignements un cours de Pensée Computationnelle pour tous ses étudiants. Dans un article de l'EPFL Magazine, nous pouvons lire :

Tout comme la physique et les mathématiques permettent d'exprimer la réalité visible sous forme d'équations rigoureuses, la pensée computationnelle la retranscrit d'une manière compréhensible pour un système informatique...permettant de résoudre des problèmes extrêmement complexes basés sur de grands ensembles de données, et de réaliser des avancées impossibles jusqu'ici...

La pensée informatique va donc permettre d'exprimer des idées en promouvant :

- La pensée abstraite
- La pensée systématique
- La pensée logique et séquentielle
- La pensée algorithmique
- La résolution de problème
- L'apprentissage par l'erreur

1.3 Qu'apporte réellement la pensée informatique aux sciences physiques ?

Si les compétences en résolution de problèmes sont les compétences les plus recherchées chez les cadres, elles sont aussi les plus difficiles à développer chez les élèves et les étudiants. Or la pensée informatique est considérée comme fondamentale en formation à la résolution de problèmes et est déjà présente en Physique Chimie. Utiliser les méthodes propres à cette pensée informatique afin de faire de la Physique et de la Chimie doit pouvoir concourir à cette compréhension profonde si difficile à réaliser pour les élèves. Enfin l'informatique occupe une place importante dans la physique et la chimie du XXI siècle. L'ignorer c'est renoncer à mettre au service de la physique et de la chimie les progrès réalisés dans ce domaine durant ces dernières années.

1.3.1 La pensée informatique pour le traitement des données

Le traitement d'informations issues d'immenses quantités de données joue un rôle de plus en plus important dans les sciences expérimentales. Comment traiter à la main les quantités phénoménales de données recueillies par un télescope ou un accélérateur de particules? L'informatique s'est intéressée depuis très longtemps à ces problèmes et propose des théories et des outils sans lesquels cette exploitation serait impossible. Ces données générées par des expériences ont été acquises grâce à l'ingéniosité des ingénieurs ou chercheurs physiciens et chimistes qui afin de pousser toujours plus loin les limites de leurs dispositifs expérimentaux se doivent de connaître les limites de ce qu'une machine peut ou ne peut pas faire. La pensée informatique utilisée dans le traitement automatisé de l'information permet aux physiciens et chimistes de tirer le meilleur parti des progrès de la science informatique afin de toujours mieux appréhender le réel. La connaissance des systèmes favorisant l'acquisition et le traitement de l'information est donc essentielle pour un physicien ou un chimiste afin que puisse être appréciées les implications de cette numérisation au service de la modélisation et de la compréhension du réel.

1.3.2 La pensée informatique nouvel outil au service de compréhension des concepts de Physique Chimie

La pensée informatique permet d'améliorer la compréhension des concepts de la physique et de la chimie tant dans l'approche expérimentale (chaine d'acquisition de l'information (capteur -> données) que dans une approche conceptuelle (modélisation / simulation) grâce à la mobilisation d'étapes usuelles en pensée informatique comme en physique chimie que sont :

- la façon de poser un problème, sa reformulation en tâches simples ;
- la prise d'initiatives en identifiant des informations ou des grandeurs importantes, afin de tester une hypothèse dans l'élaboration d'un modèle avec une démarche d'essai erreur ;
- la mise en évidence de situations qui peuvent s'apparenter à des situations déjà étudiées et des modèles déjà connus (design pattern) ;
- l'élaboration d'une solution étape par étape (algorithmic thinking) en vérifiant l'adéquation des prévisions avec la réalité.
- L'utilisation d'une représentation numérique afin de visualiser les résultats d'une simulation. Cela permet aux physiciens et aux chimistes de tester l'effet d'un paramètre ou encore d'être capable d'identifier des analogies comme les physiciens et les chimistes le font couramment.

1.3.3 Comment la mettre en place?

L'enseignement de la physique et de la chimie peut tirer un bénéfice certain d'une intégration de la pensée informatique. Simuler ce que l'on peut expérimenter, valider un modèle par l'adéquation entre les résultats obtenus par la simulation et ceux mesurés dans le monde réel, apprendre à observer le réel, affiner sa représentation du réel sont autant de pistes à explorer. La pensée pnformatique qui fait déjà partie de façon implicite de la trousse à outil du physicien et du chimiste permettrait en étant explicitée de devenir comme les mathématiques un outil clairement identifié au service du physicien et du chimiste afin d'alimenter sa réflexion critique par rapport aux problèmes rencontrés.

Nous disposons aujourd'hui de langages de programmation étudiés et utilisées par les élèves de lycée en ISN, en Mathématiques ou pour programmer à terme leurs calculatrices [1]. Facile d'accès avec des potentialités stupéfiantes [2] un langage comme Python a l'avantage de posséder des modules de calcul scientifique et une surcouche VPython moins connue mais permettant de mettre en oeuvre des objets 3D. Nous disposons également d'objets numériques comme les smartphones ou les microcontrôleurs équipant la carte Arduino et `BBC micro:bit` qui associée à des capteurs peuvent avantageusement remplacer le côté boite noire et presse bouton des centrales d'acquisition jusque-là utilisées et cela pour un coût dérisoire. Qu'on se rassure il existe même des librairies Python permettant d'utiliser certaines cartes d'acquisition présentes en lycée [3]. Des universités dont Grenoble sous l'impulsion de Joël Chevrier, Paris Diderot sous l'impusion de Julien Broboff, ou encore l'Université de Bordeaux sous l'impulsion de Ulysse Delabre ainsi que d'autres universités outre atlantique prônent une autre façon de faire de la Physique avec une approche expérimentale, résolument active de l'apprenant grâce à ces nouveaux outils numériques plus ouverts. Aux Etats Unis les travaux d'une expérimentation en école primaire d'une utilisation de la Pensée Informatique pour une approche des concepts de vitesse et d'accélération a donné lieu à une publication [4] sur la résolution de problème.

1. Certains fabricants proposent depuis peu certains modèles de calculettes programmables en Python.
2. M Ayer, Vidya & Miguez, Sheila & Toby, Brian. (2014). Why scientists should learn to program in Python. Powder Diffraction. 29. S48-D64. 10.1017/S0885715614000931].
https://www.researchgate.net/publication/269995603_Why_scientists_should_learn_to_program_in_Python
3. http://www.f-legrand.fr/scidoc/docmml/sciphys/caneurosmart/interpy/interpy.html
4. wyer, Hilary & Boe, Bryce & Hill, Charlotte & Franklin, Diana & Harlow, Danielle. (2013). Computational Thinking for Physics : Programming Models of Physics Phenomenon in Elementary School. pdf

2 ARDUINO

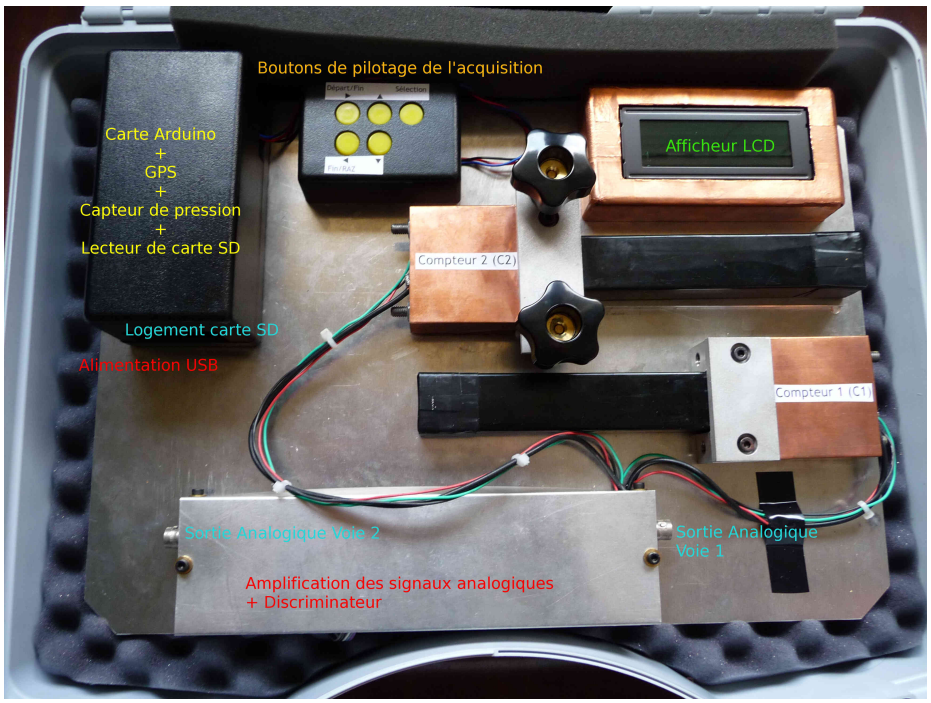
2.1 Arduino, à quoi ça sert ?

Avec Arduino on peut parler de physical computing, permettant de créer par exemple des appareils pouvant échanger de l'information avec leur environnement, grâce à des capteurs et des actionneurs dont le comportement est assuré par un programme chargé dans la mémoire de la carte Arduino. C'est une interface entre le monde analogique et le monde numérique. Les domaines d'application d'Arduino sont aujourd'hui très variés : robotique, domotique, réalité augmentée, systèmes embarqués, pilotage de capteurs pour la physique-chimie... *Avec Arduino on est limité que par son imagination !*

En complément on pourra lire :
Le making of d'Arduino ou la fabuleuse histoire d'un circuit imprimé

2.2 Arduino au lycée : le projet COSMIX

Voici un bel exemple d'utilisation d'une carte Arduino avec la malette COSMIX visant à mettre à la disposition des lycées des détecteurs de muons du rayonnement cosmique, l'information prise par le capteur est traitée par un module électronique puis envoyée à un calculateur permettant de compter les muons.



2.2.1 Présentation de la malette COSMIX par DENIS DUMORA, enseignant chercheur au CENBG

La malette COSMIX est un détecteur de particules dédié à la mise en évidence et au comptage des rayons cosmiques. Elle est construite autour de deux détecteurs de particules chargées constitués chacun d'un barreau d'Iodure de Césium instrumenté d'une photodiode et relié à une chaîne d'acquisition pilotée par une carte Arduino.

Le capteur

L'élément principal du capteur est consitué d'un barreau d'Iodure de Césium, ce sel est un matériau scintillant inorganique (il existe des scintillateurs plastiques dit "organiques"), c'est à dire un matériau qui a la propriété de produire un rayonnement dans le visible lorsqu'il est traversé par une particule chargée. Le barreau est translucide et entouré de matériau réfléchissant, la lumière produite par une particule chargée est donc guidée vers l'extrémité du cristal qui est équipé d'une photodiode. Une particule chargée traversant le barreau dépose donc de l'énergie qui est convertie par le cristal en signal lumineux visible qui vient éclairer la photodiode qui à son tour produit un signal électrique.

Les cristaux des mallettes COSMIX sont des éléments de récupération. A l'origine du projet, l'équipe du CENBG disposait de 48 *cristaux issus des tests sous faisceau des éléments du calorimètre du détecteur LAT embarqué à bord du satellite d'astronomie des rayons* γ Fermi. *Les tests effectués auprès des grands accélérateurs de particules européens (CERN, GSI, GANIL) étant terminés, les cristaux d'Iodure de Césium ont été remisés dans une armoire. L'idée a donc été de mettre ces véritables détecteurs de physique des particules à la disposition du grand public. Chacun des* 48 *barreaux a été scié en deux demi-barreaux qui constituent les deux détecteurs de la mallette COSMIX.*

Un capteur c'est quoi?

Un capteur est un appareil capable de prélever de l'information liée à une grandeur physique comme : la lumière, la chaleur, la pression, l'intensité d'une force pour la transformer en une autre grandeur physique de nature différente, très souvent électrique. Plusieurs caractéristiques du capteur sont à prendre en compte :

- *l'intervalle de mesure* : valeurs extrêmes pouvant être mesurées
- *la précision* : aptitude du capteur à donner une valeur proche de la réalité.
- *l'échantillonnage* : nombre d'informations lu durant une seconde.

Une fois connecté à votre carte arduino, un capteur nous renseigne sur le monde extérieur. On peut distinguer deux grandes familles de capteurs :

- **Les capteurs logiques (binaires)** : l'information transmise par le capteur ne peut prendre que deux valeurs : 0 ou 1. Le capteur logique le plus simple est un interrupteur. Un clavier est une matrice d'interrupteurs logiques.
- **Les capteurs analogiques** : l'information transmise est continue, très souvent proportionnelle à la grandeur physique mesurée, par exemple une photorésistance (capteur de lumière) convertit la luminosité en une valeur électrique.

La chaîne de conditionnement du signal

Le signal électrique issu de la photodiode est extrêmement faible il subit alors une première pré-amplification au niveau de l'électronique présente dans le capot à l'arrière du détecteur. Ce signal préamplifié (figure 1) rejoint alors une chaîne d'amplification et de mise en forme (figure 2) des signaux présente dans le boîtier gris à l'avant de la platine supportant les différents éléments du détecteur. Enfin, pour chacun des deux détecteurs le signal amplifié est traité par

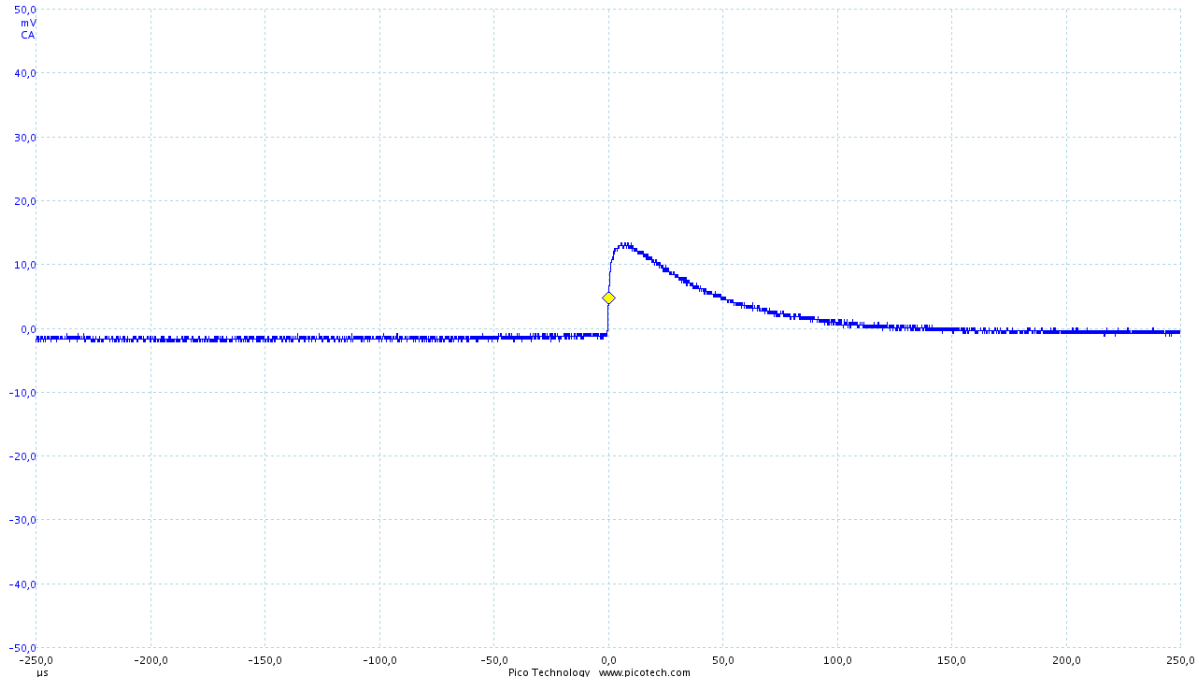


FIGURE 1 – Impulsion électrique à la sortie du préampli produite par le passage d'un muon dans le cristal de CsI. La voie de l'oscilloscope est configurée en mode AC, le signal DC sort avec un offset d'environ $-400mV$. L'axe des abscisses est en μs, celui des ordonnées en mV

un discriminateur qui génére un signal numérique TTL (état haut à $5V$) (figure 2). La coïncidence des deux signaux produit un troisième signal TTL indiquant que la particule cosmique a traversé les deux barreaux.

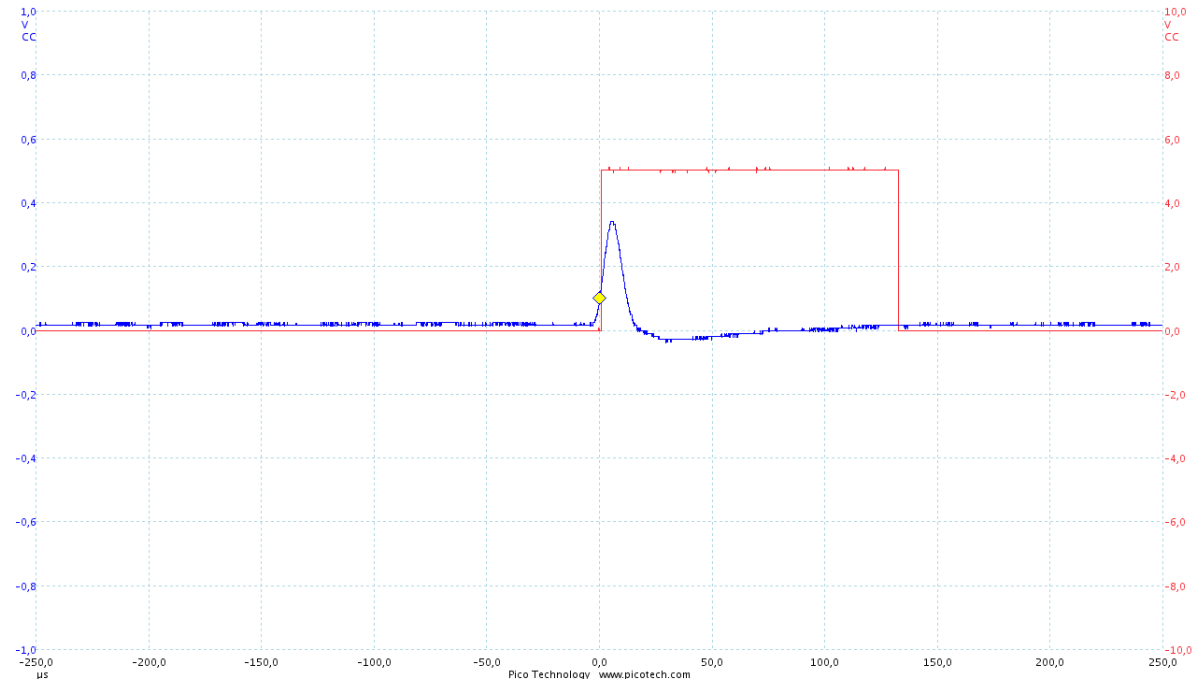


FIGURE 2 – Impulsion électrique à la sortie de l'ampli (voie A) produite par le passage d'un muon dans le cristal de CsI et signal logique TTL de déclenchement (voie B). Les voies de l'oscilloscope sont configurées en mode DC, le déclenchement de l'oscilloscope se fait sur la voie A et est positionné à $100mV$, ce qui est à peu près équivalent au seuil du discriminateur générant le signal TTL. L'axe des abscisses est en μs, celui des ordonnées en V.

Acquisition de données - La carte Arduino

Les trois signaux TTL sont alors traités par une carte Arduino MEGA, les signaux numériques issus de chacun des détecteurs sont utilisés pour déclencher une interruption dont le rôle est d'incrémenter les compteurs. A chaque interruption, le signal TTL de coïncidence est testé en lisant son état sur une voie numérique de la carte Arduino.

Autour de la carte Arduino - Interfaçage et sous-systèmes

Le micro controlleur embarqué sur la carte Arduino offre des possibilités qui vont bien au-delà de la simple capacité à compter des évènements. Dans le cadre de COSMIX, la carte Arduino va être chargée de gérer l'ensemble des fonctionnalités offertes par la mallette, de la gestion de l'interface homme machine au pilotage des séquences d'acquisition et la sauvegarde des données en passant par l'acquisition de données complémentaires en provenance d'autres sous-systèmes physiques apportant des informations complémentaires à l'acquisition de chaque évènement.

Acquisition et sauvegarde des données

La carte Arduino, est utilisée pour gérer les différentes prises de données, démarrage des comptages, arrêt des comptages, réinitialisation des compteurs software et sauvegarde des données de chaque évènement sur la carte SD

Interfaçage

La carte Arduino permet aussi la gestion des affichages sur l'écran LCD, elle gère les différents menus ainsi que l'interaction avec les boutons du panneau de commande. Elle pilote aussi les échanges sur le port série (USB) permettant ainsi d'envoyer des informations à l'ordinateur connecté et à en recevoir les instruction sur le paramétrage de l'acquisition de donnée. Un "mini" protocole est défini qui permet de transmettre à distance l'ensemble des instructions qui sont par ailleurs accessibles par le système de menu associé aux boutons du panneau de commande. Plus anecdotiquement, la carte gère aussi un système de signaux sonores permettant d'identifier quel détecteur a été touché et s'il y a eu coïncidence ou non.

Sous-systèmes

La même carte Arduino permet aussi l'acquisition sur les différents sous-systèmes connectés à la carte. Outre les systèmes d'interfaçage avec l'utilisateur et de sauvegarde des données qui ont déjà été abordés dans les sections précédentes, des modules prise de données auxiliaires sont aussi pilotés par la carte. Ainsi à chaque particule détectée, l'évènement généré contient aussi l'information de la température, de la pression (calcul de l'altitude), du temps d'arrivée (horloge RTC), et de la position GPS du détecteur.

Le code pilotant l'ensemble des fonctionalités de la mallette occupe un espace mémoire légèrement supérieur à la capacité d'une classique carte Arduino UNO, c'est pour cette raison qu'il lui a été préféré une MEGA aux capacités sensiblement supérieures en terme de stockage.

Acquérir des données avec Arduino

Même si le code de pilotage des mallettes COSMIX peut paraître impressionnant (il ne fait, malgré tout que 1520 lignes) au final le pilotage de chacune des fonctionnalités élémentaires ne demande la mise en oeuvre que de quelques fonctions, le programme complet étant pour l'essentiel construit par l'agglomération des morceaux de code relatifs à chacun des sous-systèmes, les seuls points critiques résidant éventuellement dans leur coordination.

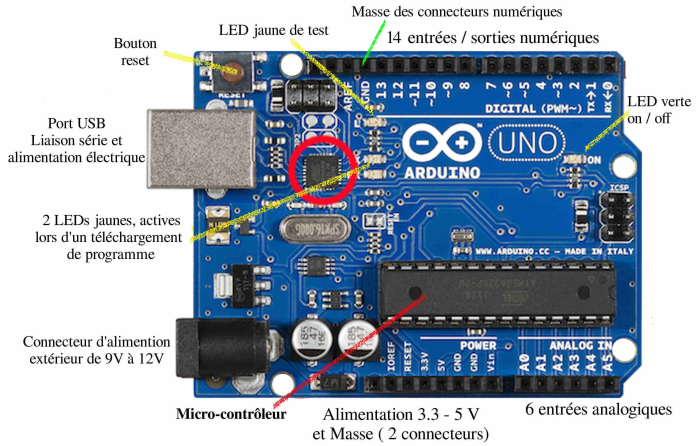
2.3 Description technique de la carte : ARDUINO UNO

C'est une carte électronique dont le coeur est un micro-contrôleur ATMEL (circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires, unités périphériques et interfaces d'entrées-sorties). de référence ATMega328. C'est un micro-contrôleur 8-bits (famille AVR) dont la programmation peut être réalisée en langage C/C++. Les micro-contrôleurs AVR embarquent dans un même boîtier :

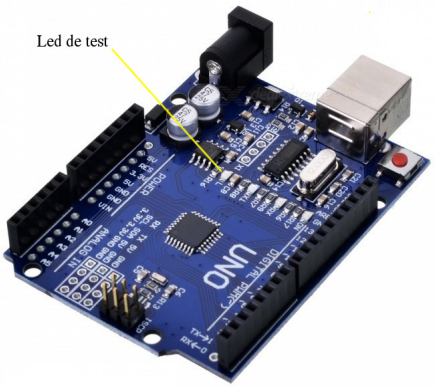
- un microprocesseur AVR,
- de la mémoire flash (espace programme),
- de la mémoire SRAM (espace données),
- de la mémoire EEPROM (espace données de sauvegarde),
- des périphériques divers.

Le tout cadencé avec une horloge à 16 MHz.

La carte officielle (≃ 20 euros)



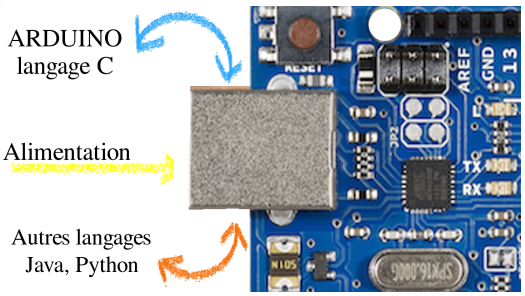
Celle du kit, un clone acheté en chine (≃ 5 euros)



Le composant entouré en rouge gère le transfert des données par la voie USB émulée en liaison série.

Vous trouverez quelques différences de positions des éléments sur la carte non officielle mais sans conséquence. Par exemple la led de test a été déplacée et marquée d'un L et la couleur n'est plus jaune mais rouge. Il ne faut pas oublier que la carte Arduino est un projet open source donc les plans sont disponibles et gratuits. La référence du microcontroleur de cette carte est : MEGA328P CH340G (16 Mhz)

Le principal intérêt des cartes ARDUINO est leur facilité de mise en oeuvre. Le chargement du programme en langage C/C++ dans la mémoire flash se fait par le port USB avec lequel il est possible de créer une liaison série pour **échanger des données avec des programmes écrits dans différents langages :** Python, Java, Processing, C, pour les plus connus.



2.4 Installation d'Arduino

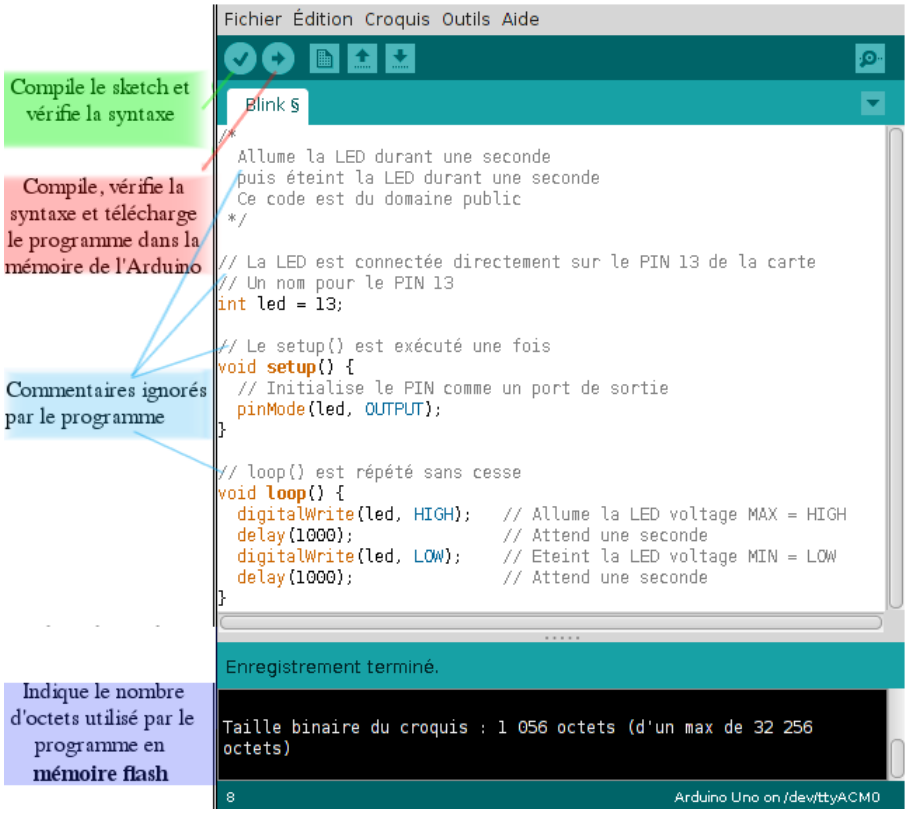
Il est très difficile d'écrire un guide d'installation complet pour tous les cas de figure possibles, j'ai donc sélectionné quelques liens qui me semblent intéressants pour l'installation du logiciel Arduino et la connexion entre votre ordinateur et la carte **Arduino UNO** :

- Windows : La documentation officielle, un blog sympa pour Windows 10, le célèbre Arduino pour les nuls, et pour finir une petite vidéo (en anglais).
- Linux : Linux,
- MacOS : macOS
- En cas de problème de driver : Installer le driver pour puce CH340 (Arduino compatible).

2.5 Le premier programme

2.5.1 Faire clignoter la diode électroluminescente servant de test sur la carte

Nous allons démarrer avec l'équivalent du fameux *Hello world* qui est traditionnellement le premier programme que l'on écrit quand on démarre la programmation. Avec Arduino il s'agit de faire clignoter la **DEL jaune de test (ou rouge suivant modèle)**. Cela permet de vérifier le bon fonctionnement et de découvrir l'environnement Arduino : liaison USB carte - PC, l'environnement de programmation encore appelé IDE ainsi que la structure d'un programme Arduino, sans faire aucun montage. Pour ce faire il suffit d'ouvrir le programme (ou sketch) `Blink` dans le menu : `Fichier` ⟶ `Exemples` ⟶ `Basics` ⟶ `Blink`.
Il se peut que le code contenu dans le fichier ne soit pas tout à fait le même, aucune importance pour le test de mise en route.



Remarque : Dans la communauté Arduino, les programmes sont appelés *sketch* en anglais, ce qui fréquemment traduit par *croquis* et que l'on peut nommer *programme*.
Une fois le programme chargé si vous pouvez observer la DEL jaune (ou orange) clignoter c'est que votre installation est réussie. Dans le cas contraire revoir la procédure d'installation.

2.6 Les notions essentielles à la programmation Arduino

2.6.1 La structure minimale d'un programme Arduino

Elle doit contenir :

- Une fonction **setup** exécutée une seule fois au lancement du programme.

  Concept de fonction : Une fonction est *un bloc* permettant d'organiser et d'éviter les répétitions de code. Elle joue un rôle important pour la modularité du code. Une fonction possède un type (void, int, float, String,...), un nom, d'eventuels paramètres entre parenthèses et un ensemble d'instructions définies dans un bloc délimité par des accolades {...}.

  Exemple : pour faire clignoter la led test, il a été nécessaire de déclarer et d'initialiser la broche numéro 13 comme une sortie numérique à l'aide de la fonction : `pinMode(13, OUTPUT)` La fonction `pinMode` accepte deux **paramètres**, le numéro de la broche et le mode (INPUT ou OUTPUT). Il existe un troisième mode INPUT_PULLUP dont je ne parlerais pas. Les valeurs passées à l'appel de la fonction sont appelées **arguments** et valent respectivement 13 et OUTPUT. Vous l'avez compris INPUT = entrée et OUTPUT = sortie.

- Une fonction **loop** dont toutes les instructions sont exécutées en boucle tant que le programme *s'exécute*. C'est grâce à cette fonction que l'on peut faire clignoter la DEL de notre programme test. Toutes les instructions associées à la fonction `loop` sont regroupées dans un bloc délimité par des accolades : { }.

  Concept d'instruction : On appelle instruction une ligne de programme qui effectue une action.

  Exemple : `digitalWrite(led, HIGH);` et `delay(1000);` sont des instructions au même titre que déclarer une variable, afficher un message à l'écran, ouvrir la liaison série ...

- Des **variables globales** permettant de déclarer les différentes broches utilisées. Elles sont déclarées en dehors de tout bloc, généralement en en-tête du programme et peuvent être utilisées dans toutes les fonctions du programme contrairement aux **variables locales** déclarées dans un bloc (fonction, if, for...) et utilisables uniquement dans ce bloc.

  Concept de variable : Une variable désigne une case (ou un ensemble de cases) de la mémoire de l'ordinateur pour stocker de l'information. Lors de la déclaration d'une variable il est impératif d'indiquer le **type** de cette variable et le **nom de variable**. Le type permet de déterminer si la variable est un entier (int), un flottant (float) ou bien une chaîne de caractères (String). Il existe beaucoup d'autres types que nous n'aborderons pas dans cette initiation.

  Exemple : `int led = 13;` est une instruction qui permet de déclarer la variable globale dont le nom est *led* de type *entier* et de l'initialiser avec la valeur 13.

Quelques exemples d'utilisation des fonctions : `setup` et `loop`

Exemple 1 :
Sur une nouvelle page : `Fichier -> Nouveau`
Tapez le programme suivant :

```
1 void setup(){
2     Serial.begin(9600);
3     Serial.print("Coucou ");
4 }
5 void loop(){
6     //vide rien à répéter
7 }
```

Exemple 2 :
Modifiez votre programme afin d'obtenir :

```
1 void setup(){
2     Serial.begin(9600);
3 }
4 void loop(){
5     Serial.print("Coucou ");
6     delay(1000);
7 }
```

1. Pour chaque exemple (à tester l'un après l'autre) téléverser, c'est à dire télécharger dans la mémoire de la carte Arduino, le programme et ouvrir le **moniteur série** (`Outils -> Moniteur série` ou bien l'icône de la loupe en

haut à droite) et en déduire l'intérêt de la fonction `loop`.

- À chaque lancement du moniteur série le programme arduino redémarre, c'est à dire qu'il est lu dans son intégralité, la fonction `setup` est de nouveau exécutée.
- L'instruction : `Serial.begin(9600);` démarre une communication série à la vitesse de 9600 bauds. Il faut donc que le moniteur série soit configuré sur ce débit... sinon vous verriez des caractères bizarres apparaitre à l'écran.

  En pratique on utilise valeur comprise entre 9600 et 115200. Plus le débit est élevé et plus la communication est rapide.

Exemple 3 :

```
1  int i;
2  void setup(){
3      Serial.begin(9600);
4      i=0;
5  }
6  void loop(){
7      String s = " mot";
8      Serial.print(s);
9      Serial.print(i);
10     i = i + 1;
11     delay(1000);
12 }
```

Exemple 4 :

```
1  void setup(){
2      Serial.begin(9600);
3      int i=0;
4  }
5  void loop(){
6      Serial.print("\t");
7      Serial.print("mot");
8      Serial.print(i);
9      i = i + 1;
10     delay(1000);
11 }
```

2. Taper l'exemple 3, téléverser le programme et ouvrir le moniteur série. Comment est affiché le contenu des variables `i` et `s` ?
3. Combien de variables sont présentes dans l'exemple 3 ? Combien y a-t-il de variables globales ?
4. Dans l'exemple 4 on obtient le message d'erreur suivant :

   ```
   sketch_aug01a.ino: In function 'void loop()':
   sketch_aug01a.ino:10:16: error: 'i' was not declared in this scope
   ```

   Quelle erreur a-t-on commise ? Modifier le programme pour qu'il n'indique plus d'erreur.
5. Quelle modification apporte l'instruction : `Serial.print("\t");` ?
   Instruction très utile dans la suite de ce document.

En complément de la structure minimale

- On se doit d'ajouter des commentaires pour une meilleur lecture des fonctionnalités du programme. Une ligne de commentaires démarre avec deux antislashs (antislash = barre oblique inversée). Il est également possible d'ajouter plusieurs lignes de commentaires en les encadrant pour commencer par une barre oblique suivie d'une étoile et pour finir d'une étoile suivie d'une barre oblique : `/* Vos commentaires */`
  Pour des exemples d'utilisation relire le programme permettant de faire clignoter la led de test.
- On peut également ajouter des fonctions autres que `setup` et `loop`. Ce que nous ne ferons pas lors de cette formation.

2.6.2 Les entrées / sorties de la carte Arduino UNO

Les entrées / sorties numériques

Sur le schéma de présentation de la carte Arduino nous avons vu qu'il y avait 14 entrées / sorties numériques et 6 entrées analogiques. Il n'y a donc pas de sorties analogiques. Pour les sorties nous utilisions la commande `digitalWrite(broche, état)`, qui est donc une commande d'écriture. Pour les entrées, nous utiliserons la commande `digitalRead(broche)`, qui vous l'aurez peut-être deviné est une commande de lecture. Une broche est donc considérée comme une entrée ou une sortie mais pas les deux. Il est donc important de bien définir si la broche va se comporter comme une entrée ou une sortie. C'est ce que nous avons fait dans l'exemple de départ *Blink* grâce à la commande : `pinMode(broche, mode)`.

- mode OUTPUT : pour indiquer à la carte que la broche doit être en mode écriture, c'est-à-dire qu'elle peut envoyer ou non du courant. C'est donc une sortie.
- mode INPUT : pour indiquer que la broche est en mode lecture. Elle ne va donc pas piloter du courant, mais être à l'écoute du courant qui lui arrive.

Pour faire clignoter la *led test* nous avons utilisé la commande `pinMode(led, OUTPUT)` pour indiquer à la broche 13 de fonctionner en mode écriture et nous avons modifié l'état (broche numéro 13) à l'aide de la fonction `digitalWrite` qui accepte deux paramètres, le numéro d'entrée/sortie de la broche et l'état HAUT ou BAS de cette broche.

Les entrées analogiques :

Pour lire les valeurs en sortie d'un capteur branché sur une entrée analogique on utilise la fonction `analogRead` qui accepte un seul paramètre, le numéro de broche. Le programme minimum est :

```
1  // On utilise la broche numéro A0
2  // const spécifie que la variable n'est pas modifiable.
3  const int broche_capteur = A0;
4
5  void setup(){
6    // Votre code
7  }
8
9  void loop(){
10   // valeur numérique lue sur la broche A0
11   // avec une variable qui n'a pas besoin d'être globale
12   int valeur_lue = analogRead(broche_capteur);
13
14   // La suite du code
15 }
```

Le résultat obtenu est une conversion de la valeur analogique en une valeur numérique réalisée par un convertisseur analogique numérique 10 bits, contenu dans le micro-contrôleur de la carte Arduino. La valeur analogique de la tension est donnée par :

Soit la valeur numérique lue :	$u_n \in [0, 2^{10}-1]$	2^{10} valeurs
Soit la tension de référence du convertisseur :	V_{ref}	tension d'alimentation de 5 V
Soit la tension analogique :	$u_a \in [0, V_{ref}]$	en volts

$$u_a = u_n \times \frac{V_{ref}}{2^{10}-1}$$

La mesure prend environ 100 μs, cela fait un maximum de 10 000 mesures par seconde.

2.6.3 L'API Arduino

Une API c'est quoi ? En français cet acronyme peut se traduire par : *interface de programmation applicative* (souvent désignée par le terme API pour Application Programming Interface). Dans notre cas cela correspond à un ensemble de fonctions natives d'Arduino dont l'utilisateur va pouvoir se servir pour écrire un programme. (Quant on utilise des langages objets comme Java ou Python le terme fonction est remplacé par *méthode*.) L'API donne une description détaillée de ces fonctions, correspondantes aux mots clés du langage de programmation, et dans le meilleur des cas un exemple.

6. À partir de l'API de référence (la même en français) lire les informations données sur la fonction `delay()`. *Pour trouver rapidement un mot dans une page web on peut utiliser un raccourci clavier en appuyant simultanément sur les touches :* Ctrl *et* F *. Une fenêtre de saisie s'ouvre généralement en bas à gauche ou à droite (ou pas!).*
   La valeur indiquée entre les parenthèses de la fonction est appelée **paramètre**. En déduire l'effet sur le programme Blink si je remplace :

```
1 void loop(){
2     digitalWrite(led, HIGH);
3     delay(1000);
4     digitalWrite(led, LOW);
5     delay(1000);
6 }
```

par

```
1 void loop(){
2     delay(500);
3     digitalWrite(led, HIGH);
4     delay(2500);
5     digitalWrite(led, LOW);
6 }
```

   Vous pouvez également avoir la liste des fonctions liées au temps au bas de la page web correspondante à la fonction `delay`
7. De même lire les informations données par l'API sur la fonction `digitalWrite`
8. Chercher une fonction permettant d'écrire un message dans le moniteur série avec un retour à la ligne. Penser à regarder du coté du module `Serial`
   Au lieu d'obtenir : `Coucou Coucou Coucou`
   On aimerait avoir :

```
Coucou
Coucou
Coucou
```

9. Écrire un programme qui permet d'afficher le numéro de la ligne en cours dans le moniteur série.

```
0 Coucou
1 Coucou
2 Coucou
...
```

10. L'installation d'Arduino permet également de récupérer la totalité de l'API sur votre support de stockage. Dans votre navigateur de fichier, faites une recherche pour accéder à l'API dans un répertoire qui devrait s'intituler `sketchbook/libraries/arduino/reference`.

2.6.4 Gestion de la mémoire de l'Arduino

Il existe trois types de mémoire sur une carte Arduino :

- **La mémoire flash** est utilisée pour stocker le programme. C'est une mémoire **non volatile**, le programme que vous avez écrit et téléversé reste dans la mémoire même hors tension.
- **La mémoire SRAM** ("Static Random Access Memory") à laquelle le programme peut accéder à tout moment. C'est là que sont copiées les données initialisées par le programme pour ensuite être modifiées. C'est une mémoire **volatile**
- **L'EEPROM** est une mémoire **non volatile** à laquelle on peut accéder par les fonctions `read()` et `write()` du package `EEPROM`. Attention ce type de mémoire est réservé aux utilisateurs initiés.

Type de mémoire	ATMega 328
Flash	32 Ko
SRAM	2048 octets
EEPROM	1024 octets

Pour toutes les cartes Arduino, 2048 octets de la mémoire Flash sont réservées pour le **bootloader** résident.

D'après Arduino - Environment

When you upload a sketch, you're using the Arduino bootloader, a small program that has been loaded on to the microcontroller on your board. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the on-board (pin 13) LED when it starts (i.e. when the board resets).

Remarque : La SRAM disponible est faible, il est préférable d'éviter les variables de type chaînes de caractères avec des chaînes trop longues. Les deux exemples ci-dessous illustrent l'espace mémoire occupé par une chaîne de caractères :

Exemple 5 :

```
1 // Un programme qui ne fait rien
2
3 String s = "";
4
5 void setup(){}
6
7 void loop(){}
```

Exemple 6 :

```
1 // Un programme qui ne fait
2 // toujours rien
3 String s = "abcd";
4
5 void setup(){}
6
7 void loop(){}
```

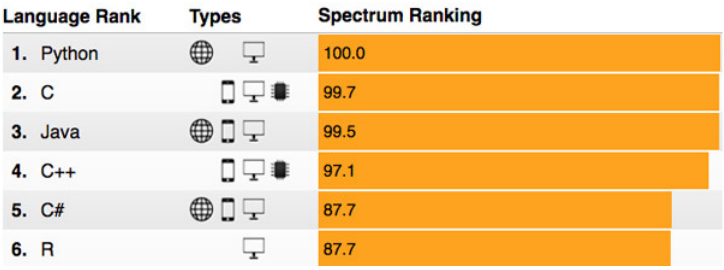
11. Copier et compiler l'exemple 5, dans la console de l'ide Arduino relever la **la taille du croquis** en octets.
12. Copier et compiler l'exemple 6, dans la console de l'ide Arduino relever la **la taille du croquis** en octets et en déduire l'espace mémoire occupé par un caractère de la chaîne en octets.

3 Le couple Arduino - Python

3.1 Pourquoi Python ?

3.1.1 Avantages pour les scientifiques

- Facile à installer, libre et multi-plateformes (Linux, Windows, macOS)
- Prise en main très rapide (quelques jours)
- Alternative fiable à des logiciels spécialisés (matlab, excel, libreOffice...)
- Spécialisé dans le calcul scientifique, la représentation des données sous forme de graphiques et la simulation
- Utilisation simple de la liaison série pour le transfert de données avec Arduino
- Python est un des langages les plus populaires d'après L'Institute of Electrical and Electronics Engineers (IEEE) qui est la plus grande association mondiale de professionnels techniques (IEEE Spectrum). Un article intéressant à lire à ce sujet : IEEE : Python devient le meilleur langage en 2017 en dépassant C et Java



3.1.2 Avantages pour les élèves

- Un document ressources de l'éducation nationale mentionnant clairement Python, vient de paraître pour la rentrée 2017 sur le thème algorithmique et programmation
- Python est très majoritairement utilisé dans l'enseignement de spécialité ISN en terminales S.
- Python est un enseignement obligatoire en C.P.G.E depuis la rentrée 2013
- L'informatique avec Python est une épreuve obligatoire des concours aux grandes écoles que ce soit sous forme d'une épreuve de simulation pour la physique - chimie (Concours communs polytechniques et e3a) ou d'une épreuve d'informatique pour tous plus théorique (Centrale-Supélec et Mines-Ponts)

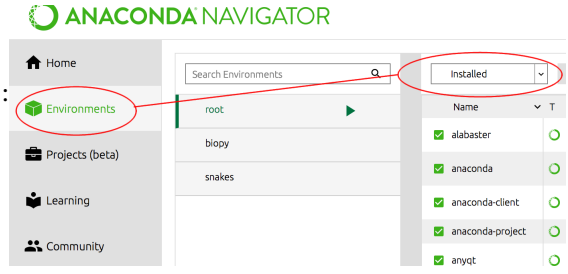
3.2 Installation de Python

Préparer la formation : Téléchargement d'Anaconda 3.xx puis Installation sur le site officiel en anglais. Il semble que les versions récentes d'Anaconda **ne contiennent pas forcement** le package **pyserial** dont nous allons avoir besoin pour communiquer avec Arduino (À tester lorsque vous aborderez l'exemple : *Lecture des données* 3.4.2).

3.2.1 Installation du package pyserial

Si lors du test vous obtenez une erreur avec le package *pyserial*, vous pouvez l'installer de différentes manières :
depuis l'interface d'Anaconda Navigator (recommandé)

- Cliquer dans la fenêtre d'accueil sur le menu : **Environments**.
- Un menu déroulant en haut de la fenêtre indique par défaut : **installed**
- Choisir **Not installed**
- Sélectionner dans la liste le module **pysérial**.
- Valider pour l'installation.



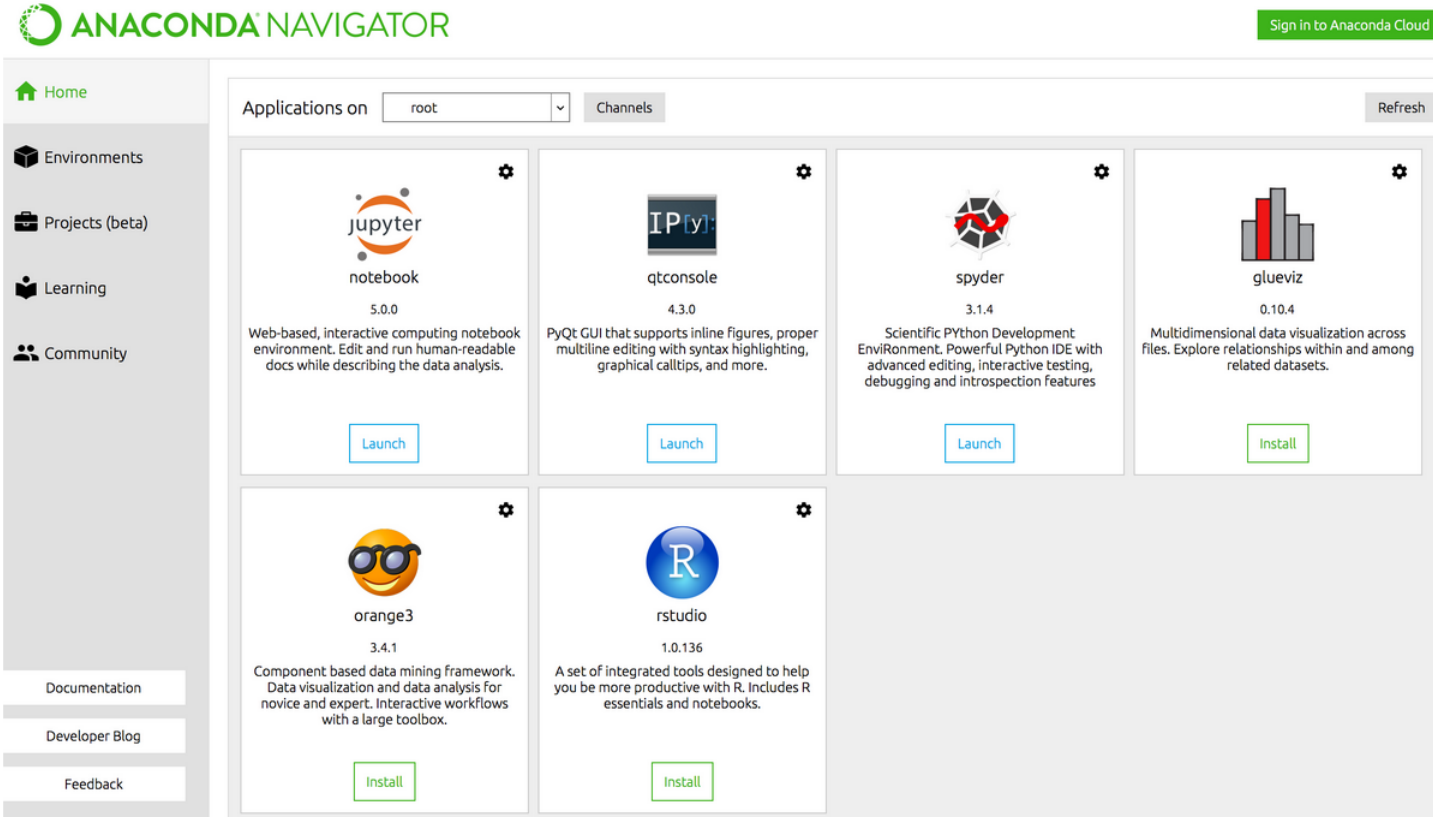
depuis Conda ou PyPI :

- Linux / macOS / Windows : PySerial

3.3 Utilisation du Jupyter Notebook

Maintenant que Python est installé sur votre ordinateur, il nous faut un environnement de programmation. Pour faire très simple, un éditeur de texte, permettant d'écrire et d'interpréter du code Python (et bien plus...). Pour cela nous allons utiliser le Jupyter Notebook. Jupyter est une application web utilisée pour programmer dans plus de 40 langages de programmation. Jupyter permet de réaliser des notebooks, c'est-à-dire des feuilles de programmes contenant à la fois du texte (en markdown), du code Python et pour les connaisseurs vous pourrez même insérer du code LaTeX pour rédiger de belles équations. Ces notebooks sont très utilisés en science pour explorer, analyser et présenter des données. Exemple de notebook pour le test d'un capteur infrarouge.

Pas de panique le Jupyter Notebook est présent dans la distribution Anaconda que vous venez d'installer. Elle propose un bureau graphique appelé Anaconda Navigator. Il ne reste plus qu'à lancer Jupyter Notebook avec le bouton **Launch**.



Pour bien démarrer voici un petit guide Jupyter Notebook pour en savoir un peu plus sur le Jupyter Notebook.

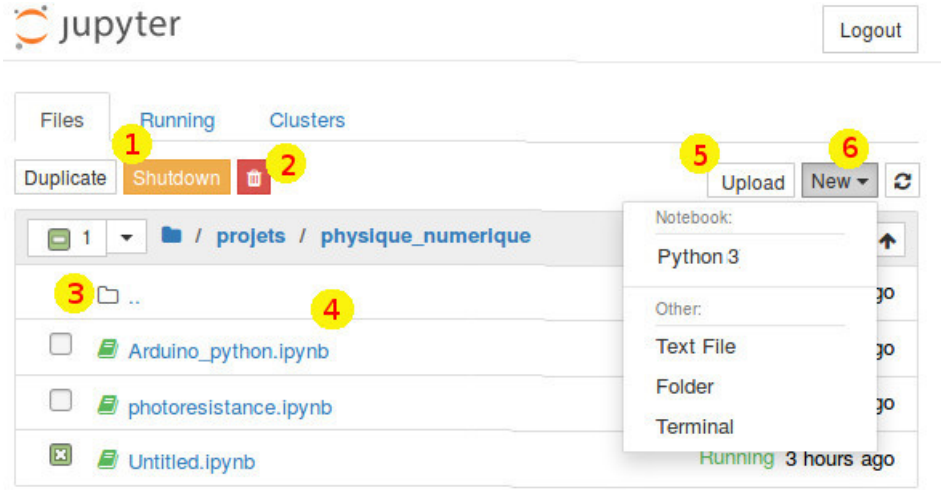
Remarques :

- Quand vous exécutez le programme *Jupyter Notebook*, une fenêtre avec un fond noir s'affiche à l'écran, elle permet d'observer les commandes qui lancent les services (noyau Python, navigateur web, ...) du *Jupyter Notebook*. Surtout ne pas la fermer.

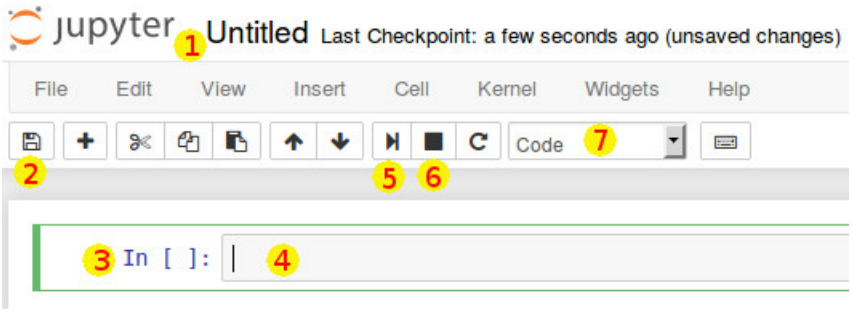
- Si tout se passe bien Jupyter lance un navigateur web servant d'interface graphique pour la programmation Python. Il se peut que la première fois le lancement soit relativement long. Un jour une élève est venue me voir pensant que l'installation n'avait pas marché. Après un rapide état des lieux de sa machine, j'ai constaté qu'il fallait plusieurs minutes à Windows pour lancer son antivirus (Avast) ce qui décalait d'autant le lancement d'Anaconda Navigator. Donc dans certains cas patience...

Vous pouvez également travailler à partir d'une version en ligne : try.jupyter. Attention cette version en ligne ne permet pas d'utiliser le package `pyserial`. Mais elle reste tout de même très performante pour travailler avec des élèves.

3.3.1 Description sommaire de l'interface web du notebook



1. Fermer un notebook
2. Effacer un notebook
3. Dossier parent
4. Liste des notebooks, cocher pour sélectionner un notebook
5. Charger un notebook
6. Créer un nouveau notebook avec Python 2 ou 3 suivant les versions



1. Clic gauche pour changer le titre (Untitled) du notebook.
2. Sauvegarder le notebook
3. Cellule du notebook
4. Zone de code python
5. Exécuter le code (ou *shift* + *enter*)
6. Stopper l'exécution du code
7. Sélection du type de contenu dans la cellule en cours.

3.4 Communication Arduino - Python via le port série

3.4.1 Dans quel but ?

La carte Arduino permet de faire l'acquisition d'un signal analogique par l'intermédiaire d'un capteur et de le convertir en signal numérique grâce à son CAN (10 bits). Il est ensuite possible de transférer ces données par le port série vers l'ordinateur pour réaliser un traitement numérique avec Python.

Capteur	**Echantillonage et conversion**	**Traitement numérique du signal**
Photorésistance	Carte d'acquisition Arduino	Python

Donc très schématiquement on se sert de l'interface de programmation d'Arduino pour écrire un petit programme qui explique à la carte comment faire l'acquisition (programme qui est ensuite téléverser sur la carte par le port série) puis on récupère les données via le port série pour en faire une analyse avec Python.

3.4.2 Lecture des données envoyées par la carte Arduino avec Python

Le code Arduino ci-dessous envoie une valeur entière aléatoire toutes les secondes en passant à la ligne après chaque valeur (réponse à la question 8).

Remarque : *If it is important for a sequence of values generated by random() to differ, on subsequent executions of a sketch, use randomSeed() to initialize the random number generator with a fairly random input, such as analogRead() on an unconnected pin.*

Conversely, it can occasionally be useful to use pseudo-random sequences that repeat exactly. This can be accomplished by calling randomSeed() with a fixed number, before starting the random sequence.

Code Arduino à téléverser sur la carte

```
1 void setup(){
2     Serial.begin(9600);
3     randomSeed(analogRead(0));
4 }
5 void loop(){
6     Serial.println(random(1,100));
7     delay(1000);
8 }
```

Code Python : Pour notre premier exemple, nous allons créer une liaison série pour que Python puisse communiquer avec la carte Arduino :

- **Fermer le moniteur série** coté Arduino, pour pouvoir établir une liaison avec Python
- Ouvrir un nouveau Notebook.
- Changer le nom du notebook : Arduino_Python
- Recopier l'exemple ci-dessous en n'oubliant pas d'exécuter la cellule de code. Attention de bien indiquer le port sélectionné dans le menu Arduino (`Outils -> Port série`). Sous Windows : COM suivi d'un numéro (1, 2, 3, ...), sous linux : /dev/ttyACM suivi d'un numéro (0 ou 1 en général) ou /dev/ttyUSB0

```
1 import serial
2 serial_port = serial.Serial(port = "COM1", baudrate = 9600)
3 serial_port.readline()
```

j'ai effectué une copie d'écran afin que l'on puisse avoir une vue du programme dans le Jupyter Notebook. Chaque case est ce que l'on appelle **une cellule** et l'ensemble des deux cellules forme notre programme Python. Les cellules ne sont pas indépendantes les unes des autres, **elles forment un tout**, comme si **le code avait été écrit dans une seule et même cellule**. Attention lors de la copie d'écran j'ai séparé les lignes 1 et 2 de la ligne 3 afin d'introduire quelques commentaires.
Le résultat de notre programme peut être visualisé sur la sortie standard (`out[3]:` dans le notebook) avec un nombre entier aléatoire suivi de 4 caractères indiquant la fin de ligne et le retour à la ligne, ces caractères peuvent changer en fonction du système d'exploitation : `'35\r\n'`.

Il y a également deux cellules de texte pour donner quelques explications. La mise en forme d'une cellule de texte se fait à l'aide de la syntaxe Mardown

Dans cet exemple pour obtenir une nouvelle valeur, il faut relancer à chaque fois la cellule contenant l'instruction : `serial_port.readline()`. Pour afficher plus de valeurs on peut utiliser une **structure de contrôle** appelée boucle. Si on veut dix valeurs on peut écrire :

```
1 for i in range(10):
2     print(serial_port.readline())
```

Attention le fait de valider plusieurs fois une cellule pour obtenir de nouvelles valeurs n'effectue pas un reset de la carte Arduino, contrairement au fait de fermer le moniteur série (cf 2.6.1). C'est à dire que le setup n'est pas relu. Pour s'en convaincre il suffit de tester le programme suivant :

Code Arduino à téléverser

```
int i;
void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(0));
  i = 0;
}
void loop(){
  Serial.print(i);
  Serial.print("\t");
  Serial.println(random(1,100));
  i = i + 1;
  delay(1000);
}
```

Coté Python il n'y a rien à changer. Vous pouvez éventuellement modifier le nombre d'acquisitions. Si vous exécutez plusieurs fois le code Python vous vous apercevrez que **le compteur des valeurs transmises n'est jamais remis à zéro**.

Pour relancer la fonction *setup* du code Arduino vous devez fermer la liaison série avant d'en ouvrir une nouvelle.

Code Python pour relancer la fonction *setup* sur Arduino

```
serial_port = serial.Serial( port = "COM1", baudrate =9600)
for i in range(10):
    print(serial_port.readline())
serial_port.close()
```

Si on veut pouvoir observer une bonne synchronisation, c'est à dire récupérer les premières valeurs transmises par Arduino à partir de la réinitialision du microcontroleur et ce quelque soit la vitesse d'acquisition et de transmission, on peut utiliser le code Python suivant :

```
serial_port = serial.Serial( port = "COM1", baudrate =9600)
# réinitialisation
serial_port.setDTR(False)
time.sleep(0.1)
serial_port.setDTR(True)
# on vide le buffer
serial_port.flushInput()
# lecture des données
for i in range(10):
    print(serial_port.readline())
serial_port.close()
```

Une des broches matérielles de contrôle du flux (DTR) du circuit intégré ATmega est connecté à la ligne de réinitialisation de l'ATmega 328 via un condensateur de 100 nanofarads. Lorsque cette broche est mise au niveau BAS, la broche de réinitialisation s'abaisse suffisamment longtemps pour réinitialiser le microcontrôleur. On force la réinitialisation juste avant la lecture des données envoyées par l'Arduino.

4 Exemples d'utilisation de la carte Arduino en physique

4.1 La carte Arduino Uno un système d'acquisition

Parmi les nombreuses façons d'utiliser la carte Arduino, une des possibilités est d'en faire un système d'acquisition permettant au physicien ou au chimiste de récolter à l'aide d'un ou plusieurs capteurs les informations essentielles à l'analyse d'une expérience. Dans ce contexte la carte Arduino peut-être utilisée comme :

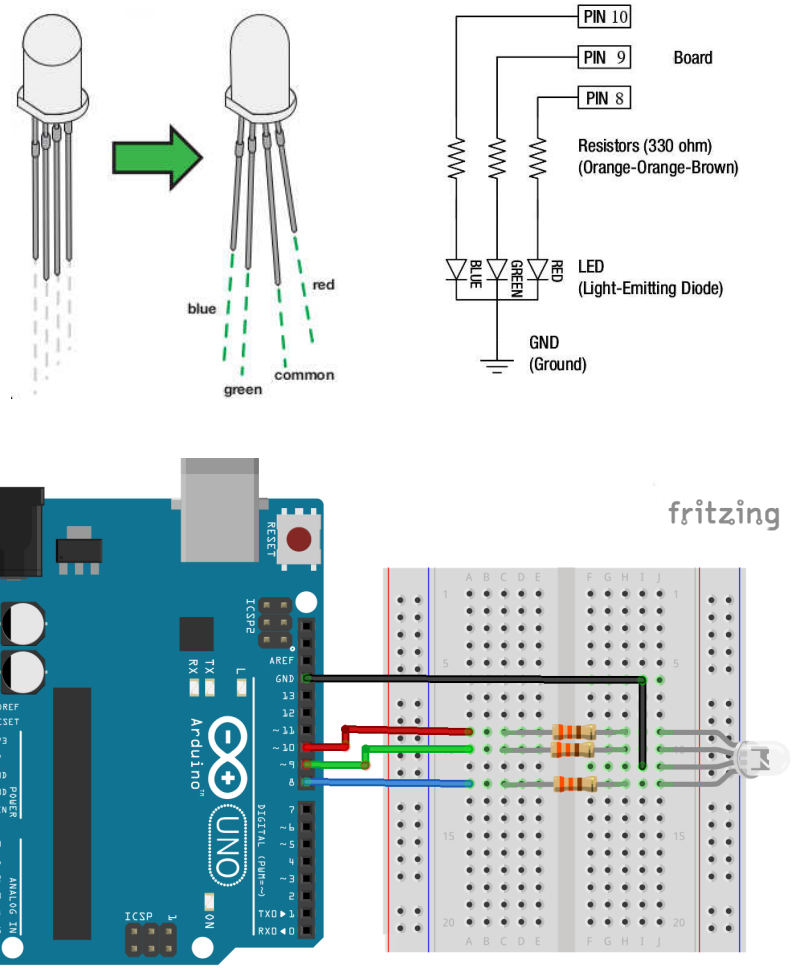
- serveur de données (data server) vers un langage de programmation à travers une liaison série ou USB.
- enregistreur de données (datalogger) autonome avec un shield SD, les données sont sauvegardées sur une carte SD pour ensuite être lues et exploitées à l'aide d'un langage de programmation ou d'un logiciel.
- pseudo système embarqué (embedded system) avec traitement des données, des opérations de calcul doivent être faites en réponse à un évenement extérieur avant stockage ou transmission de l'information.

Il est également possible de piloter des actionneurs à partir des sorties numériques. Certaines sorties numériques peuvent simulées une sortie analogique avec la modulation de largeur d'impulsion (pulse width modulation ou PWM). Il est ainsi envisageable de réaliser par exemple la caractéristique d'un dipôle résistif.

4.2 Synthèse additive des couleurs avec un actionneur de type diode électroluminescente RGB

Dans un premier temps, l'objectif est d'utiliser une DEL capable de produire trois couleurs différentes et de réaliser une synthèse additive avec les couleurs rouge, vert et bleu.

4.2.1 Les schémas électriques (Documentation Arduino)



4.2.2 La réalisation

13. À partir du programme test (cf 2.5), écrire un programme Arduino permettant de faire clignoter la LED avec une fréquence de 1 Hz et avec une alternance des couleurs Rouge, Vert, Bleu.
14. Modifier votre programme pour obtenir le clignotement de la LED mais avec l'alternance de couleurs Jaune, Cyan, Magenta et Blanc.
15. Combien de couleurs peut-on obtenir en utilisant la fonction `digitalWrite`?
16. Il est possible d'augmenter le nombre de couleurs grâce à la fonction : `analogWrite`. Lire l'API Arduino pour comprendre l'utilisation de cette fonction.
17. Écrire un programme Arduino permettant d'obtenir quatre niveaux d'intensité différents sur un même canal (0, 75, 125, 250) avec un changement toutes les secondes. Attention il faudra peut-être modifier la position de votre DEL suivant vos attentes. En effet seules les broches précédées d'un ~ sont utilisables avec la fonction : `analogWrite`
18. Combien de couleurs peut-on obtenir avec la fonction : `analogWrite`?

4.3 Étude statistique de la mesure d'une tension avec la carte Arduino Uno

4.3.1 Meilleur estimation de la moyenne et de l'écart type d'une distribution de valeurs

La meilleure estimation de la valeur vraie d'une grandeur X , notée $\bar{x}$ est donnée par la moyenne arithmétique définie comme suit :

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Par suite, la meilleure estimation de l'écart type sur la population des x_i est donnée par :

$$\sigma_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Dans l'hypothèse où toute erreur systématique a été ecartée et où les mesures individuelles x_i sont reparties selon une loi normale on appelle incertitude type élargie la valeur donnée par :

$$\Delta(x) = t_{n,p\%} \times \frac{\sigma_x}{\sqrt{n}} = t_{n,p\%} \times u(x)$$

Avec $t_{n,p\%}$ coefficient de Student donné par les tables et $u(x)$ incertitude type.

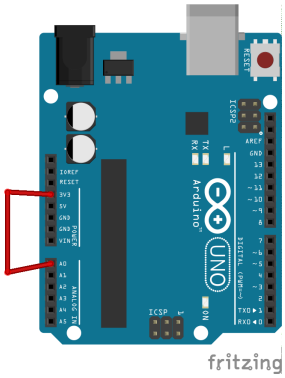
L'intervalle de confiance s'exprime sous la forme : $[\bar{x} - \Delta x; \bar{x} + \Delta x]$

Les résultats des mesures effectuées de la grandeur X sont ensuite présentés sous la forme :

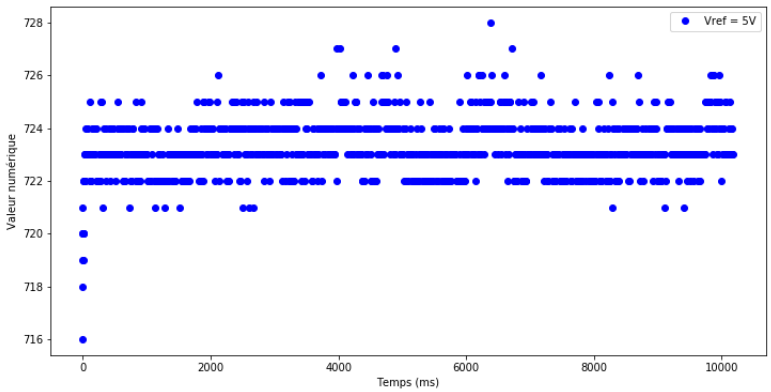
$$X = \bar{x} \pm \Delta x$$

4.3.2 Effet de la discrétisation lors de l'acquisition

Le montage suivant va nous permettre d'étudier l'effet de la discrétisation d'une tension analogique à travers le convertisseur analogique-numérique (CAN) de la carte Arduino Uno.

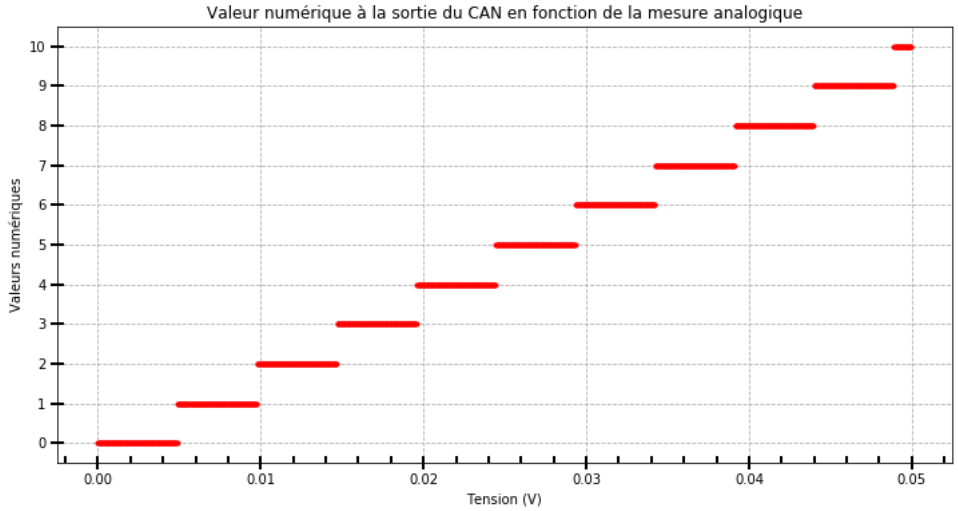


19. À l'aide du chapitre sur les entrées / sorties de la carte Arduino, écrire un programme Arduino permettant d'obtenir sur la liaison série les valeurs numériques de cette tension en fonction du temps.
20. Modifier le code pour obtenir exactement 100 mesures.
21. Effectuer un copier-coller de vos valeurs dans un logiciel permettant de réaliser le graphique suivant. Les logiciels libres Veusz et LabPlot peuvent être un bon compromis à des logiciels plus connus mais souvent plus onéreux.



On constate que les valeurs prises par la tension mesurée sont discrètes. Cette discrétisation est due à la conversion analogique numérique du CAN de la carte Arduino Uno. De manière générale si les valeurs analogiques sont codées avec une **précision sur n-bits** alors nous disposons de 2^n valeurs possibles dans l'intervalle $[0, 2^n - 1]$. En d'autres termes il faut bien prendre conscience que toutes les valeurs analogiques de l'intervalle $[0, V_{ref}]$ **ne seront pas exactement représentables** par les valeurs numériques. Il est intéressant de noter que le plus petit écart entre deux valeurs numériques représente **le pas en tension** encore appelé **résolution** noté par la suite ∂u

$$\partial u = \frac{V_{ref}}{2^n}$$



22. Déterminer la résolution associée à la tension 3.3 V de la carte Arduino Uno sachant que d'après les spécifications constructeur $V_{ref} = 5\,V$ et que $n = 10$.
23. Repérer sur le graphique ci-dessus quelques intervalles de tension dont la longueur correspond à celui du pas en tension.
24. Quelle est la valeur numérique d'une mesure dans l'intervalle $[0\,;\,4.8\,mV]$?
25. À quel intervalle en tension peut-on associer la valeur numérique 7 ?
26. En déduire la valeur moyenne de la tension délivrée par la broche 3.3 V

4.3.3 Mesure avec la carte Arduino Uno et loi normale

La mesure d'une tension constante engendre un grand nombre de valeurs identiques mais nous observons également une certaine fluctuation de cette valeur sans doute due à l'état électronique du circuit qui n'est pas rigoureusement le même dans le temps et à une erreur aléatoire de mesure. Intéressons nous à cette erreur aléatoire et essayons de vérifier qu'elle suit bien une loi normale.

Pour cela on mesure une tension $U \simeq 3.3\,V$ (toujours avec le montage précédent) en réalisant une acquisition de $N = 1000$ points. On peut alors effectuer un traitement statistique de l'erreur aléatoire basé sur la répétition des mesures de la tension u. Dans notre cas il s'agit d'extraire les meilleurs estimations de la valeur moyenne de u notée $\bar{u}$ et de l'écart type σ_u.

4.3.4 Récupération et stockage des données avec Python

Nous allons maintenant utiliser Python pour établir une liaison USB destinée au transfert des données de Arduino vers Python, puis pour stocker les données reçues en vue d'un traitement ultérieur. Pour cela il faut commencer par ouvrir un nouveau notebook que l'on pourra renommer : discretisation, **attention pas d'accent dans les noms de Notebook**

Dans la première cellule de code recopier les importations des packages nécessaires à la gestion de cet exemple.

Code Python

Les déclarations de packages.

```
1 import serial                    # gestion du port série
2 import matplotlib.pyplot as plt  # pour faire de beaux graphiques
```

Dans une deuxième cellule nous allons nous concentrer sur l'écriture du code principal permettant de récupérer et d'organiser les données. Attention la connexion Python avec le port série demande à être adaptée en fonction de votre système d'explotation (3.4.2).

```
1  # connexion Linux au port série
2  serial_port = serial.Serial( port = "/dev/ttyACM0", baudrate = 9600)
3  # les mesures
4  mesures = []
5  temps   = []
6  serial_port.flushInput()
7  for i in range(1000):
8      val = serial_port.readline().split()
9      try:
10         t = float(val[0])
11         m = float(val[1])
12         mesures.append(m)
13         temps.append(t)
14     except:
15         pass
16 # fermeture du port série
17 serial_port.close()
```

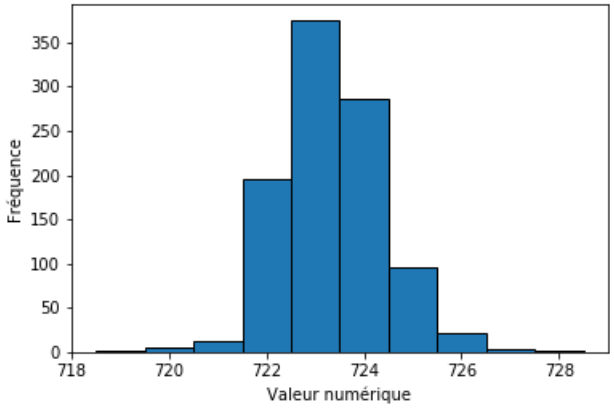
- lignes 4 et 5 : déclaration de deux listes qui recevront les mesures de l'expérience (comme dans les colonnes d'un tableur).
- ligne 6 : On vide la mémoire tampon de la liaison série. Cela permet d'effacer d'éventuelles données de l'acquisition précédente restées en mémoire.
- ligne 7 : On démarre une boucle qui permet de récupérer 1000 valeurs. Toutes les instructions associées à la boucle sont indentées.
- ligne 8 : permet de lire le flux de données envoyé sur le port série par Arduino. Rappelez-vous que le programme Arduino envoie deux valeurs sur le port série, le temps et la mesure du capteur. Il faut donc séparer ces deux valeurs. Pour cela nous utilisons la fonction `split()`. Elle sépare les valeurs grâce à l'espace que nous avons laissé et les range dans une liste Python. Une liste Python peut-être vue comme un tableau dont chaque case porte un numéro.
  - La première case a le numéro 0.
  - Pour ajouter une valeur dans la liste on utilise la fonction `append()`
  - Pour lire la valeur contenue dans la première case de la liste `val` on écrit : `val[0]`, pour lire la valeur contenue dans la n ième case on écrit : `val[n]`, pour lire les valeurs comprises entre les cases n et m incluses on écrit : `val[n : m+1]`
- ligne 9 : **try** permet de gérer une erreur d'exécution dans un programme sans pour autant que le programme s'arrête brutalement.
  Le mécanisme de gestion des exceptions s'effectue en deux phases :
  - La levée d'exception avec la détection d'erreurs : le problème se trouve lignes 10 et 11 lors de la conversion.
  - Le traitement approprié : ligne 15 nous décidons de ne rien faire avec le mot clé **pass**
- ligne 10 à 13 : Nous essayons de convertir les données reçues en valeurs décimales et nous les ajoutons aux listes `temps` et `mesure` . N'oublions pas que les données envoyées par Arduino sont au format texte. Il est donc nécessaire de les convertir en valeur numérique. La conversion réalisée est de type *float* soit des valeurs décimales.
- ligne 14 et 15 : Si cela ne marche pas, on passe et on lit une nouvelle ligne envoyée par Arduino
- ligne 17 : surtout **ne pas oublier de fermer le port de communication**

Normalement plus de mystère vous pouvez maintenant recopier le code correspondant à l'acquisition des mesures. Le gros avantage c'est que l'on écrit le code une fois pour toute. Il peut même être déja disponible dans un notebook que vous donnez à vos élèves. Mais rien n'empêche de le modifier pour satisfaire une demande particulière. Par exemple on peut vouloir enregistrer les données de manière persistante sur le disque dur ou bien sur une carte SD.

4.3.5 Affichage des données sous forme d'histogramme

```
1 # l'intervalle des mesures doit être adapté avec vos valeurs
2 plt.hist(mesures, range=[718.5, 728.5], bins=10, edgecolor = 'black')
3 plt.xlabel("Valeur numérique")
4 plt.ylabel("Fréquence")
5 plt.show()
```

L'histogramme des mesures est représenté ci-dessous :



4.3.6 Traitement des données : moyenne, écart type et loi normale

En vue du calcul de la moyenne et de l'écart type de la distribution nous allons ajouter à la cellule des packages, le package `scipy`. Remarquer la différence de syntaxe avec l'ajout de la totalité d'un package.

```
1 from scipy import std, mean
```

En fait nous n'ajoutons pas toutes les fonctionnalités de ce module mais seulement celles dont nous avons besoin c'est à dire `std` pour l'écart type et `mean` pour la moyenne.

Dans une nouvelle cellule il ne reste plus qu'à calculer moyenne et écart type

```
1 moyenne = mean(mesures)
2 std_m   = std(mesures, ddof=1)
3 print("Moyenne numérique = ", moyenne, "\nÉcart type numérique = ", std_m)
```

```
Moyenne numérique = 723.891
Écart type numérique =  1.4727929250237457
```

On peut également ajouter la représentation graphique de la fonction de *distribution normale* définie par :

$$f : x \mapsto \frac{1}{\sigma_x \sqrt{2\pi}} e^{\frac{-(x-\bar{x})^2}{2\sigma_x^2}} \qquad \text{où } \bar{x} = \text{ moyenne et } \sigma_x = \text{ écart type}$$

La surface totale comprise entre la courbe de la fonction f et l'axe des x vaut 1. Pour tracer la fonction f appliquée à notre exemple il est donc nécessaire que l'aire sous la courbe de f dans l'intervalle étudié soit égale à l'aire des rectangles de l'histogramme. L'expression de la fonction f devient :

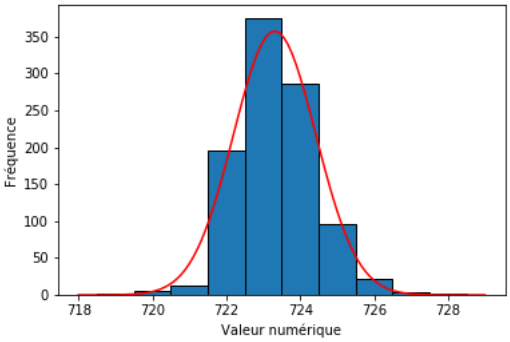
$$f : x \mapsto \frac{A}{\sigma_x \sqrt{2\pi}} e^{\frac{-(x-\bar{x})^2}{2\sigma_x^2}}$$

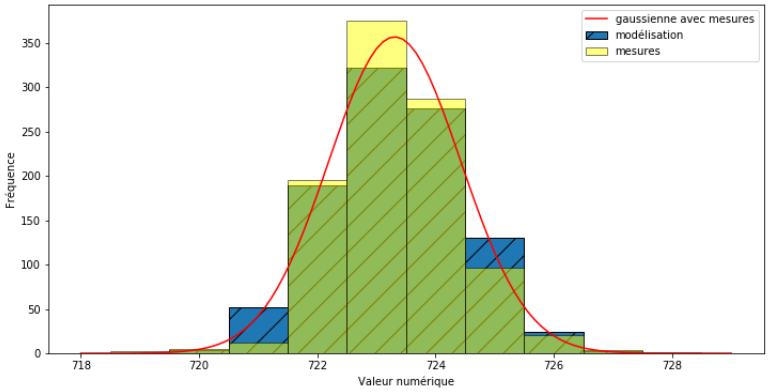
A représente l'aire sous la courbe que l'on calcule avec la valeur de l'intervalle de classe et le nombre d'échantillons soit : $A = 1000 * 1$
Avoir une *distribution normale* des échantillons, cela signifie que 95% des mesures les plus proches de la moyenne sont dans un intervalle dont la largeur est de : $4\sigma_x \simeq 6$

$$x = 724 \pm 3$$

Avec Python

```
1  import numpy as np
2
3  # La fonction de la loi normale
4  def gaussienne(x, A, moyenne, ecarttype):
5      return A/(ecarttype*np.sqrt(2*np.pi))*np.exp(-(x-moyenne)**2/(2*ecarttype**2))
6
7  #Intervalle de définition
8  x = np.linspace(min(mesures), max(mesures),100)
9  y = Gaussienne(x, 1000, moyenne, std_m)
10
11 #Le graphique
12 plt.hist(mesures, range=[718.5, 728.5], bins=10, edgecolor = 'black')
13 plt.xlabel("Valeur numérique")
14 plt.ylabel("Fréquence")
15 plt.plot(x,y,'red')
16 plt.show()
```





Les graphiques ci-dessus montrent que la loi de distribution de la mesure d'une tension délivrée par la carte Arduino Uno est **approximativement une loi normale**, c'est à dire que la probabilité de trouver la grandeur X dans l'intervalle $[\bar{x}-2\sigma_x, \bar{x}+2\sigma_x]$ est de 95%. Ce qui permet d'appliquer le principe d'incertitude élargie à 95%.
Dans l'hypothèse où toute erreur systématique a été écartée et où les N mesures individuelles x_i sont réparties selon une loi normale

$$\Delta x = t_{(N,p\%)}\frac{\sigma_x}{\sqrt{N}} \qquad \text{avec } t_{(N,p\%)} \text{ le coefficient de student}$$

27. Réaliser l'histogramme de droite avec les informations suivantes.

```
1  """
2  On peut afficher deux histogrammes superposés en appliquant un effet
3  de transparence (alpha) sur le dernier histogramme affiché.
4  """
5
6  #... à compléter
7  plt.hist(x1, bins = bins, edgecolor = 'red',
8           hatch = '/', label = 'x1')
9  plt.hist(x2, bins = bins, color = 'yellow', alpha = 0.5,
10         edgecolor = 'blue', label = 'x2')
11 #...
12 plt.legend()
13 plt.show()
```

```
1  # génération d'un tableau de points tirés au hasard suivant une loi normale
2  import numpy as np
3  n = 1000                                  # nombre de points
4  x = np.random.normal(moyenne, std_m, n) # tableau de points
```

28. Réaliser la même étude statistique en remplaçant l'alimentation Arduino 3.3 V par une alimentation stabilisée et comparer moyenne et écart type.
29. À l'aide des documents ci-dessous étudier l'influence de la tension de référence sur la carte Arduino Uno pour une mesure de tension $\simeq 0.8\,V$

Tension de référence de la carte Arduino

Le micro-contrôleur de l'Arduino possède plusieurs tensions de référence utilisables selon la plage de variation de la tension que l'on veut mesurer.... Pour cela, rien de matériel, tout se passe au niveau du programme car il existe une fonction `analogReference`. Cette fonction prend en paramètres le nom de la référence à utiliser.

- DEFAULT : La référence de 5 V par défaut.
- INTERNAL : Une référence de 1.1 V sur la carte Arduino Uno
- EXTERNAL : La référence est celle appliquée sur la broche AREF

Comment utiliser la fonction `analogReference`?

```
1  void setup(){
2
3    // permet de choisir une tension de référence de 1.1V
4    analogReference(INTERNAL);
5  }
```

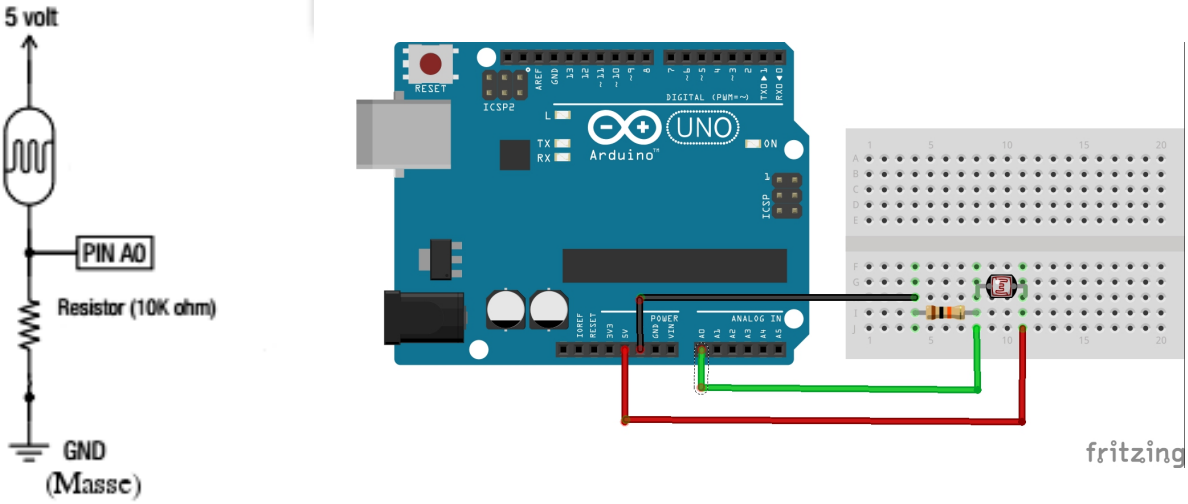
4.3.7 Des idées pour la suite

- On peut compléter cette exemple en mesurant une même tension, d'une part avec la carte Arduino Uno avec une méthode statistique et d'autre part avec un multimètre numérique. En récupérant la précision du multimètre sur la notice on peut exprimer l'intervalle de confiance de la mesure et ainsi comparer les résultats. Il est alors envisageable de mettre en évidence une éventuelle erreur systématique si les intervalles de confiance sont disjoints.
- L'incertitude type évoluant en $\frac{1}{\sqrt{N}}$ avec N le nombre de mesures, on peut aussi se questionner sur l'évolution de la largeur de l'intervalle de confiance quand le nombre de mesures augmente.

4.4 Mesure de fréquence avec capteur analogique de type photorésistor ou photodiode

4.4.1 Le montage électrique

D'après la doc Arduino



4.4.2 Mesurer la fréquence d'un stroboscope (application smartphone)

Deux possibilités s'offrent à nous. Soit on utilise un stroboscope classique que l'on trouve normalement au laboratoire de physique ou bien il peut être remplacé par une application smartphone.

- Télécharger n'importe quelle application de stroboscope sur votre smartphone, pour cette expérience j'ai utilisé Stroboscopique
- ou télécharger l'application Physics Toolbox Suite, puis cliquer sur stroboscope dans le menu de gauche.
- Régler la fréquence sur 1 Hz
- Placer le flash de votre téléphone au dessus de la photorésistance ou de la photodiode

Le code Arduino associé à cette expérience est extrêmement simple, il nous suffit juste de lire les valeurs envoyées par le capteur (photorésistance ou photodiode) sur une des entrées analogiques de la carte Arduino. Rappelez-vous, celle-ci propose 6 entrées analogiques (de A0 à A5) connectées à un convertisseur analogique-numérique 10 bits (2^{10} valeurs possibles de 0 à $2^{10}-1$). Ce qui permet de transformer la tension d'entrée comprise entre 0 et 5V en une valeur numérique entière comprise entre 0 et 1023. La résolution (écart entre 2 mesures) est de : 5 volts / 2^{10} intervalles, soit 0.0049 volts (4.9 mV).

30. Compléter puis téléverser le code Arduino.
31. Comment faudrait-il modifier le code pour que le nom de variable `valeur` référence bien une tension (attention au type de la variable).

Code Arduino

```
// Variables à déclarer

void setup(){
  Serial.begin(19200);
}

void loop(){
  // À compléter            // valeur numérique lue sur la broche A0
  Serial.print(valeur);   // Envoi la mesure au PC par la liaison série (port USB)
  Serial.print("\t");       // Ajout d'un espace
  Serial.println(millis()); // Envoi de la valeur temps puis retour à la ligne
                            // Une éventuelle temporisation
}
```

32. À l'aide du moniteur série, observer les résultats obtenus.
33. À l'aide d'un tableur, comment tracer le graphique correspondant à $u = f(t)$?

Nous allons maintenant utiliser Python pour automatiser la gestion des données envoyées par le capteur. Pour cela il faut commencer par ouvrir un nouveau notebook que l'on pourra renommer : stroboscope

Dans la première cellule de code recopier les importations des packages nécessaires à la gestion de cet exemple.

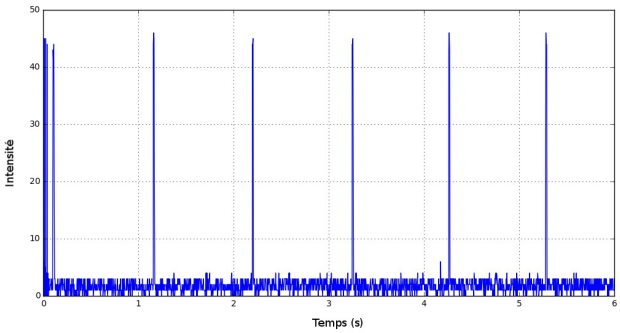
Code Python

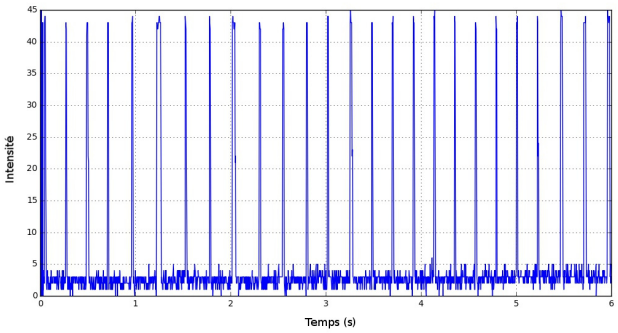
34. Écrire le code Python permettant d'utiliser les instructions suivantes pour l'affichage du résultat.

L'affichage sous la forme d'un graphique

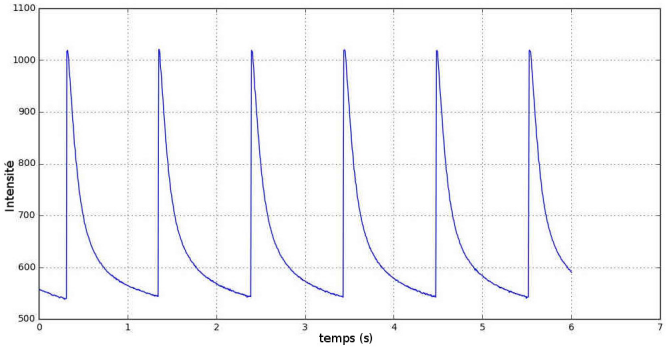
```
# On évite les effets de bord en éliminant
#les valeurs de début et de fin de transmission
plt.plot(temps[100:900], mesure[100:900])
plt.xlabel("Temps (s)")
plt.ylabel("Intensité")
plt.grid()
plt.show()
```

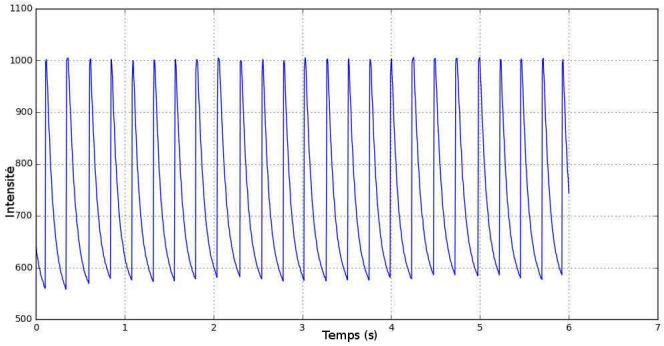
Photodiode (f = 1Hz et 4Hz)





Photorésistance (f = 1Hz et 5Hz)





On voit qu'après chaque flash (supposé suffisamment court), le photorécepteur reste conducteur pendant une durée qui va dépendre du type de photorécepteur

- pour la photodiode temps de réponse très court de quelques microsecondes. Cela illustre la bonne réponse en fréquence de la photodiode.
- pour la photoresistance un temps de réponse relativement court mais elle reste conductrice durant plusieurs dixièmes de secondes

On peut améliorer la lecture du flux de données afin d'assouplir l'utilisation du code Python. Pour cela nous allons écrire deux fonctions Python dont nous détaillerons l'utilisation.

```
def acquisition(n, serial_port):
    '''
    Cette fonction permet de faire l'acqusition des données
    en fonction du temps reçues par le port USB.
    Elle renvoie deux listes :  temps et mesures (du capteur)

    n           <int>    : nombre total de valeurs à lire
    serial_port <serial> : le port série ouvert à la communication
    '''
    i = 0
    temps, mesures = [], []
    while i < n:
        val = serial_port.readline().split()
        try:
            t = float(val[1])
            m = float(val[0])
            temps.append(t)
            mesure.append(m)
            i = i + 1
        except:
            pass
    return temps, mesures
```

35. Comment le code de la fonction `acquisition` a t-il été modifié par rapport au code précédent et pourquoi?

Pour lancer une acquisition avec 1 000 points :

```
# connexion Linux au port série
serial_port = serial.Serial( port = "/dev/ttyACM1", baudrate =19200)
serial_port.setDTR(False)
time.sleep(0.1)             # attention le module time est nécessaire
serial_port.setDTR(True)
serial_port.flushInput()

temps, mesure = acquisition(1000, serial_port)

# fermeture du port série
serial_port.close()
```

4.4.3 Fixer la durée d'acquisition

Dans l'exemple précédent l'acquisition dépend d'un nombre de points. Mais il est souvent plus utile de pouvoir contrôler le temps d'acquisition. Le code Arduino ne change pas et le code Python ne va subir qu'une toute petite modification au niveau de la boucle. Au lieu de compter un nombre de points nous allons définir un temps d'acquisition. Rappelons que le code Arduino transmet à chaque itération de la fonction `loop` une ligne contenant une **valeur** et une **date** d'acquisition. Pour contrôler le temps d'acquisition il suffit donc de surveiller la différence entre la date en cours d'acquisition et la date du début d'acquisition. Comme les dates d'acquisition sont dans une liste `temps`, nous allons surveiller `temps[-1] - temps[0]` avec :

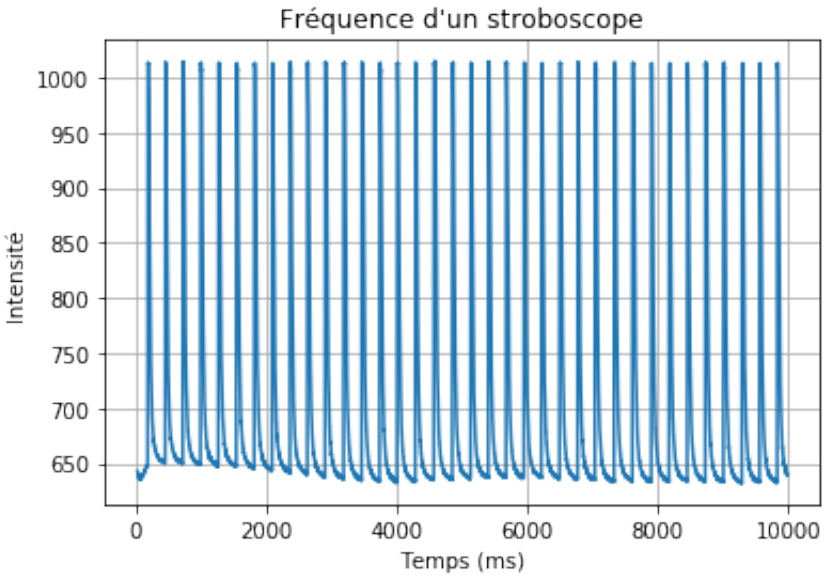
- `temps[-1]` le dernier élément de la liste `temps`
- `temps[0]` le premier élément de la liste

```
# ouverture du port série
serial_port = serial.Serial( port = "/dev/ttyACM0", baudrate =19200)
serial_port.setDTR(False)
time.sleep(0.1)
serial_port.setDTR(True)
serial_port.flushInput()

# les mesures
mesure = []
temps = []
duree = 10000
end = False

while end == False or temps[-1] - temps[0] <= duree:
    val = serial_port.readline().split()
    try:
        t = float(val[1])
        m = float(val[0])
        temps.append(t)
        mesure.append(m)
        end = True
    except:
        pass
# fermeture du port série
serial_port.close()
```

36. Écrire une fonction `acquisition_temps(duree, serial_port)` qui prend en paramètres la durée d'acquisition et la connexion au port série. Cette fonction renvoie dans l'ordre la liste des dates et mesures de l'acquisition.

L'affichage sous la forme d'un graphique

```
# attention les deux listes doivent contenir le même nombre de valeurs.
plt.plot(temps, mesure)

plt.title("Fréquence d'un stroboscope")
plt.ylabel('Intensité')
plt.xlabel('Temps (ms)')
plt.grid()
plt.show()
```



4.5 Utilisation d'un bouton poussoir pour déclencher l'acquisition

L'objectif est **d'ajouter à l'expérience du stroboscope, un bouton poussoir** pour déclencher l'acquisition coté Arduino afin que Python puisse enregistrer les données transférées. Dans cet exemple, très fortement inspiré d'une activité de Jean-Luc Charles [5], nous parlerons **d'automate**.

Concept d'automate

Un automate fini est un modèle mathématique des systèmes ayant un nombre fini d'états et que des actions (externes ou internes) peuvent faire passer d'un état à un autre.

Rien de mieux qu'un exemple pour comprendre :

- à l'état initial, l'automate est à l'état WAIT : l'acquisition est en attente,
- l'appui sur le bouton poussoir fait passer l'automate dans l'état START : l'acquisition démarre,
- un nouvel appui sur le bouton poussoir fait passer l'automate de l'état START à l'état STOP : l'acquisition est suspendue,
- les appuis successifs font passer successivement de l'état START à l'état STOP, et de l'état STOP à l'état START.

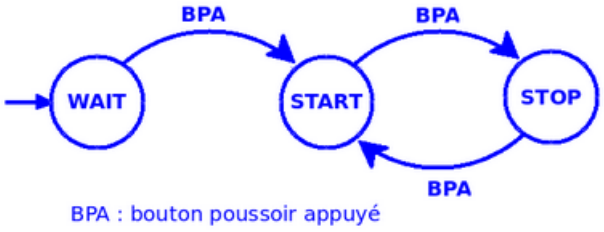
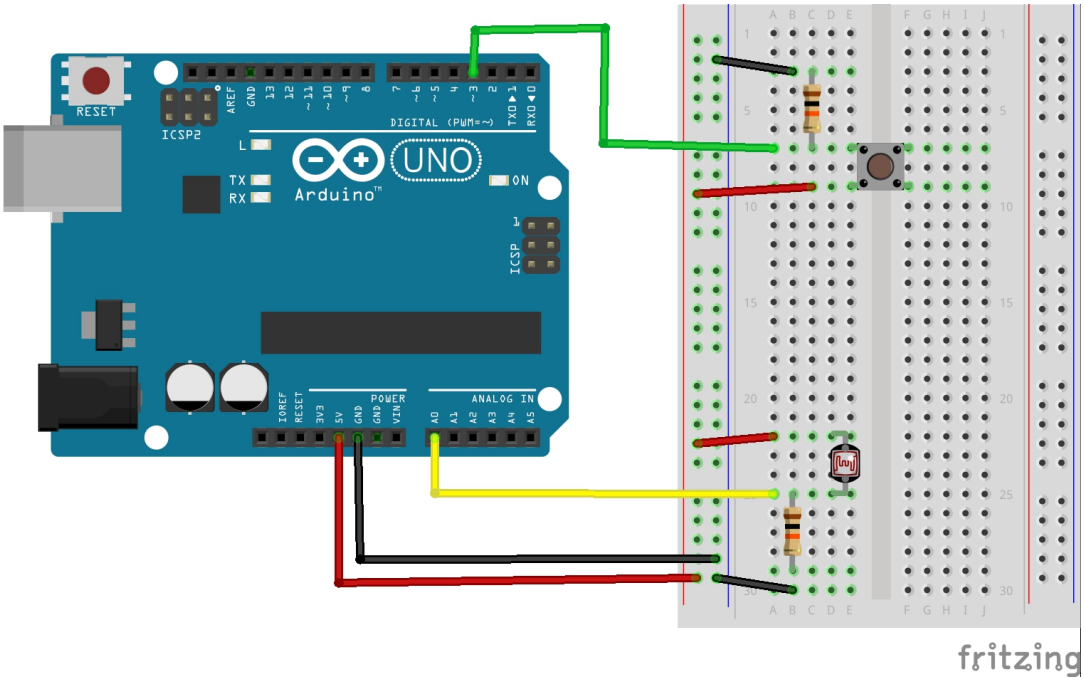


Image extraite d'une activité de Jean-Luc Charles (note : 5)

4.5.1 Le montage électrique



La broche numérique 3 de la carte Arduino est utilisée comme une entrée numérique qui reste à LOW tant que le bouton n'est pas enfoncé. Le bouton se comporte comme un intérrupteur qui ne laisse pas passer le courant tant qu'il est en position haute. Dans cet exemple la broche 3 est en mode INPUT : `pinMode(3, INPUT)`, pour indiquer que la broche est en mode lecture. **Elle ne va donc pas piloter du courant, mais être à l'écoute du courant qui lui arrive.**

5. Jean-Luc Charles, enseignant chercheur à l'ENSAM Talence

4.5.2 Le code Arduino

Coté Arduino ça donne quoi? Commençons par les variables globales et la fonction `setup`

```
1  // Etat en cours de l'automate
2  int etat;
3  // Etat à mémoriser
4  int oldEtat;
5
6  //Les états possibles de l'automate
7  const int WAIT = 2;
8  const int START = 1;
9  const int STOP  = 0;
10
11 // Les broches utilisées
12 //capteur
13 const int broche = A0;
14 //bouton poussoir
15 const int BP = 3;
16
17 void setup() {
18   //initialisation des variables
19   oldEtat = LOW;
20   etat = WAIT;
21   //config E/S
22   pinMode(BP, INPUT);
23   //liaison série
24   Serial.begin(19200);
25 }
```

Comme convenu dans l'introduction nous avons défini les différents états de notre automate et initialisé une variable `oldEtatBP` qui nous permettra de garder en mémoire l'état du bouton avant un nouvel appui. On remarquera également que l'état de notre automate est `WAIT`, nous attendons le démarrage de l'acquisition.

```
1  void loop() {
2    int etatBP = digitalRead(BP);            // Lecture du bouton
3    if(oldEtat == LOW && etatBP == HIGH){    //gestion des états
4      if (etat == WAIT)
5      {
6        etat = START;
7      }
8      else if (etat == STOP)
9      {
10       etat = START;
11     }
12     else if (etat == START)
13     {
14       etat = STOP;
15     }
16   }
17   //Traitement des états
18   if(etat == START){
19     int valeur = analogRead(broche);
20     Serial.print(valeur);
21     Serial.print("\t");
22     Serial.println(millis());
23   }
24   oldEtat = etatBP;
25   delay(10);
26 }
```

Il n'y a plus qu'à tester le programme :

- Téléverser le programme dans la mémoire de la carte Arduino.
- Ouvrir le moniteur série.
- Lancer l'acquisition en appuyant une première fois sur le bouton
- Stopper l'acquisition en appuyant une deuxième fois sur le bouton.
- On peut recommencer autant de fois que l'on veut...

37. Modifier le programme pour que lorsque l'acquisition s'arrête, c'est à dire que l'on appuie sur le bouton pour la deuxième fois, la chaîne `-1\t -1` s'affiche dans le moniteur série. Attention dans le moniteur série le `\t` sera remplacé par une tabulation.

4.5.3 Le code Python

Attention le code Arduino ci-dessous fonctionnera correctement uniquement si vous avez répondu à la question précédente, si cela pose un problème consulter la solution dans les annexes, compléter le code Arduino et poursuiver.

```
# les modules à importer
import serial
import matplotlib.pyplot as plt
```

```
# ouverture du port série et synchronisation des données entre arduino et Python.
serial_port = serial.Serial( port = "/dev/ttyACM0", baudrate =19200, timeout = None)
serial_port.flushInput()

# les mesures
mesure = []
temps = []
end = False

while end == False:
    val = serial_port.readline().split()
    if val[0] == b'-1':                    # Bouton poussoir à l'état STOP
        end = True                         # Terminaison de la boucle
    else:
        try:
            t = float(val[1])
            m = float(val[0])
            temps.append(t)
            mesure.append(m)
        except:
            pass

# fermeture du port série
serial_port.close()
```

Pour tester l'ensemble, assurez-vous que vous avez bien effectué les étapes de la section précédente coté Arduino :

- Valider les cellules du Notebook.
- Normalement sur la gauche de la deuxième cellule vous observez une petite étoile : **In [*]**
- Pas de panique avec le bouton on a tout notre temps.
- Positionner votre stroboscope au dessus de la photorésistance
- Lancer l'acquisition des valeurs en appuyant une première fois sur le bouton.
- Terminer l'acquisition en appuyant une deuxième fois sur le bouton, si tout c'est bien passé l'étoile de votre **In [*]** disparait pour laisser place à un nombre.
- Afficher vos résultats dans un graphique.

À chaque fois que l'on termine une acquisition il faut revalider la cellule du notebook contenant le code ci-dessus pour mettre en attente le code Python d'une nouvelle acquisition. L'instruction `serial_port.close()` réinitialise le code Arduino et met donc l'automate coté Arduino dans l'état `WAIT`. Il n'y a plus qu'à appuyer sur le bouton...

4.6 Temps de réponse d'une thermistance de type CTN

4.6.1 Présentation de la thermistance CTN

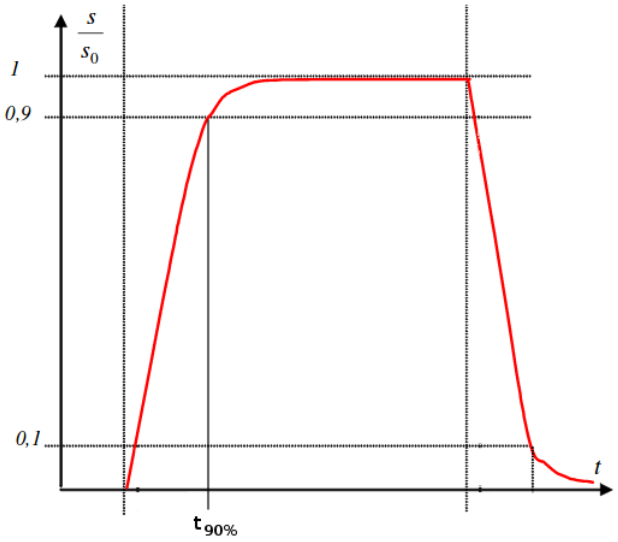
La dépendance en température d'une résistance CTN n'est pas linéaire, elle peut être bien approximée en fonction de la résistance R_{Th} de la thermistance à l'aide de la relation suivante :

$$\theta = \frac{1}{\frac{\ln\left(\frac{R_{Th}}{R_0}\right)}{\beta} + \frac{1}{T_0}} - 273.15$$

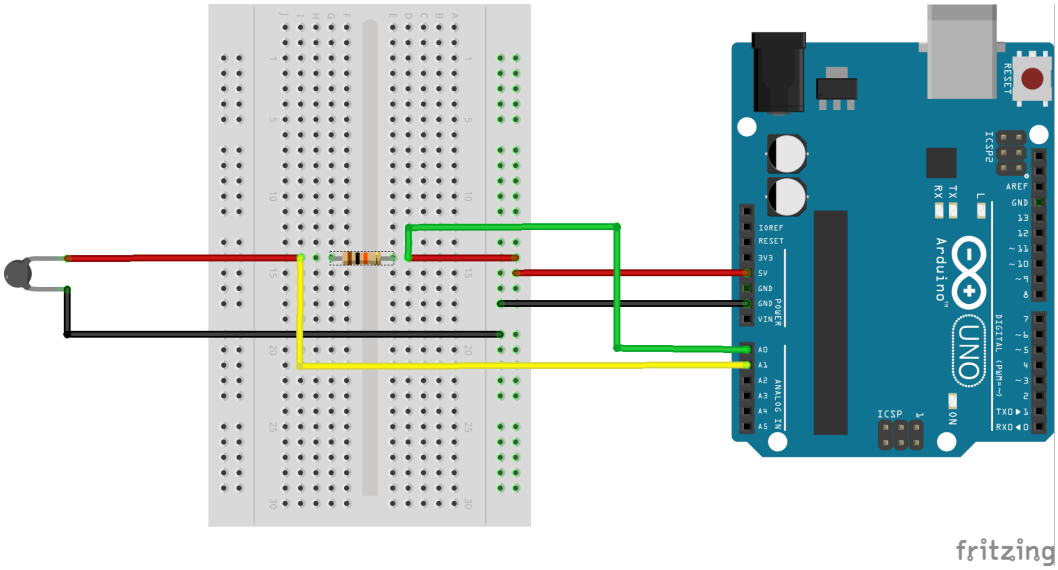
β est une constante de température (kelvin), R_0 est une résistance de référence (ohms) et T_0 est la température à laquelle s'applique la résistance de référence (kevlin). Ces constantes sont caracteristiques de la thermistance utilisée.

Le temps de réponse d'un capteur en température n'est pas nul car le capteur ne s'adapte jamais instantanément à une variation de température du milieu ambiant (air, eau, huile moteur, ...). Si la température du milieu ambiant passe d'une valeur θ initiale à une valeur θ finale supérieure à la température initiale θ initiale , le temps de réponse $t_{90\%}$ est la durée nécessaire pour que la température mesurée par le capteur passe de la valeur θ initiale à la valeur :

$$\theta = \theta_{\text{initiale}} + 0,9 \times (\theta_{\text{finale}} - \theta_{\text{initiale}})$$



4.6.2 Le montage électrique



Il est possible d'acheter des thermistances CTN (10K) étanches pour un prix très raisonnable (< 1 € ref : NTC Thermistance précision capteur de température 10 K 1% 3950 Sonde Étanche 1 m)

- $\beta = 3380\,\text{K} \pm 1\%$
- $R_0 = 10\,\text{k}\Omega \pm 1\%$
- plage de mesure :-20 à 105 ^{o}C avec $T_0 \simeq 298\,\text{K}$

4.6.3 Les codes du montage

Coté Arduino

Le code à compléter ci-dessous peut-être l'occasion de discuter de la conversion d'une valeur numérique codée sur 10 bits (A0 et A1) en valeur analogique. Pour compléter ce code nous pourrons utiliser la fonction `map`.

La fonction `map` avec Arduino

Ré-étalonne un nombre appartenant à un intervalle de valeurs vers un autre intervalle de valeurs. Dans notre cas la valeur numérique $\in [0,1023]$ en valeur analogique $\in [0\,\text{V}, 5\,\text{V}]$

`map (valeur, limite_basse_source, limite_haute_source, limite_basse_destination, limite_haute_destination)`

- valeur : le nombre à ré-étalonner
- limite_basse_source : la valeur de la limite inférieure de la fourchette de départ
- limite_haute_source : la valeur de la limite supérieure de la fourchette de départ
- limite_basse_destination : la valeur de la limite inférieure de la fourchette de destination
- limite_haute_destination : la valeur de la limite supérieure de la fourchette de destination

Il est possible d'obtenir des informations supplémentaires et des exemples liés à cette fonction avec l'API d'Arduino.

```
1  //Fonction setup(), appelée au démarrage de la carte Arduino
2  void setup()
3  {
4    Serial.begin(9600); // initialisation de la communication série à 9600 bps
5  }
6  // Fonction loop(), appelée en boucle si la carte Arduino est alimentée
7  void loop()
8  {
9    // Declaration des variables
10   unsigned long temps=millis();
11   double U, Uther, tensionU, tensionUther;
12   // lecture des tensions U et Uther sur A0 et A1 et initialisation
13   // du compteur temporel
14   temps = millis()/1000;
15   U = analogRead(0) ;
16   // conversion de la tension lue sur A0 en valeur analogique
17   tensionU =
18   // conversion de la tension lue sur A0 de mV en V
19   tensionU =
20
21   Uther = analogRead(1) ;
22   // conversion de la tension lue sur A1 en valeur analogique
23   tensionUther =
24   // conversion de la tension lue sur A1 de mV en V
25   tensionUther =
26
27   // Envoi les mesures sur le port série
28   Serial.print(tensionU);
29   Serial.print("\t");
30   Serial.print(tensionUther);
31   Serial.print("\t");
32   Serial.println(temps);
33
34   // intervalle de temps d'une seconde (1000 ms) entre
35   // deux executions de la  boucle donc entre deux mesures
36   delay(1000);
37 }
```

Coté Python

```
1 # Cellule 1 : les modules à importer
2 import serial
3 import time
4 import numpy as np
5 import matplotlib.pyplot as plt
6 %matplotlib auto
```

Dans la cellule des modules à importer rien de nouveau à part la ligne 6. Dans les exemples précédents, nous avions l'habitude d'écrire `%matplotlib inline`. Nous avons remplacé `inline` par `auto` afin de pouvoir afficher une fenêtre extérieure au Notebook. Cette fenêtre offre quelques fonctionnalités de base étendues comme la possibilité de suivre la courbe avec un réticule.

La cellule suivante donne la définition d'une fonction permettant d'effectuer la synchronisation temporelle pour le transfert des valeurs entre Arduino et Python. Les instructions liées à cette fonction ont déjà été décrites dans la partie *Communication Arduino - Python via le port série*

```
1  # Cellule 2
2  def synchro_arduino(port_name, vitesse):
3      """
4      Cette fonction initialise et renvoie une référence sur la connexion
5      avec la carte arduino à travers le port série (USB).
6      Elle permet également de synchroniser le lancement de l'acquisition
7      coté Arduino avec la récupération des données coté Python.
8      """
9      serial_port = serial.Serial( port = port_name, baudrate =vitesse)
10     serial_port.setDTR(False)
11     time.sleep(0.1)
12     serial_port.setDTR(True)
13     serial_port.flushInput()
14     return serial_port
```

À vous de compléter la fonction `modelisation_CTN` afin de calculer la température correspondante aux mesures reçues par Python.

```
1  # Cellule 3
2  def modelisation_CTN(mesures):
3      """
4      Cette fonction réalise le traitement des données, associées au capteur
5      thermistance, reçues de la carte Arduino.
6      Elle renvoie :
7          tensionU     -> float : la tension délivrée par la carte Arduino
8          tensionUther -> float : la tension aux bornes du capteur
9          Rther        -> float : la valeur de la résistance de la thermistance
10         temps        -> float : la date de la mesure
11         temperature  -> float : la valeur de la temperature
12     Elle prend en argument la liste des mesures effectuées par Arduino
13         tensionU     -> float
14         tensionUther -> float
15         temps        -> float
16     """
17     Rzero = 10000  # en ohms
18     beta  = 3380   # en Kelvins
19     Tzero = 298    # en Kelvins
20
21     tensionU, tensionUther, temps = mesures
22
23     Rther       = tensionUther*(1/(tensionU-tensionUther)*Rzero)
24     temperature = # À compléter
25
26     return tensionU, tensionUther, Rther, temps, temperature
```

La dernière cellule concerne essentiellement l'affectation des valeurs à afficher à la bonne structure de données, dans notre cas des listes Python. Cette cellule est pratiquement identique à celle des exercices précédents sans le bouton poussoir. Libre au lecteur de l'adapter s'il en ressent le besoin. J'ai juste ajouté le formatage des données pour une bonne lecture dans la sortie du Notebook. J'ai essayé de faire en sorte que cela ressemble à un tableau. On peut faire bien mieux en utilisant un module plus adapté comme Pandas avec ses DataFrames mais cela sortirait du cadre de cette formation.

```
1  # Cellule 4
2  # ouverture du port série
3  serial_port = synchro_arduino("/dev/ttyACM0", 9600)
4
5  # les mesures
6  temperature = []
7  temps       = []
8  duree       = 100     # en seconde (1min 40s)
9  end         = False
10
11 # On s'occupe de l'affichage des résultats
12 head = "tensionU\t tensionUther\tRther\ttemps\ttemperature\n"
13 print(head.expandtabs(10))
14 fmt = "{0:.2f}\t{1:.2f}\t{2:.2f}\t{3:.2f}\t{4:.2f}".expandtabs(16)
15
16 # on récupère les données et on modélise
17 while end == False or temps[-1] - temps[0] < duree:
18     data_arduino = serial_port.readline().split()
19     try:
20         mesures = np.array(data_arduino, dtype='float') # String -> flottant
21         mesures = modelisation_CTN(mesures)             # Calcul température
22         temps.append(mesures[3])            # remplissage liste des temps
23         temperature.append(mesures[4])      # remplissage liste des températures
24         print(fmt.format(*mesures))         # formatage de l'affichage
25         end = True
26     except:
27         pass
28 # fermeture du port série
29 serial_port.close()
```

4.6.4 L'expérience et ses résultats

De nombreuses expériences sont possibles, pour ma part j'ai déposé quelques glaçons dans le fond d'un premier récipient avec un peu d'eau puis j'ai rempli un deuxième récipient avec un peu d'eau à la température de la pièce.

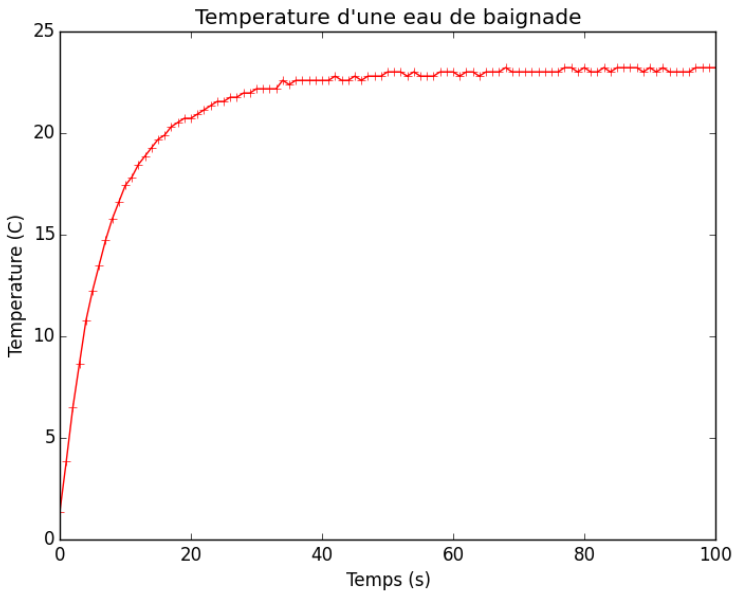
- Téléverser le code Arduino dans la mémoire de la carte
- Valider les cellules 1, 2 et 3
- Plonger la thermistance environ 30 secondes dans le récipient avec les glaçons
- Plonger la thermistance dans l'eau du deuxième récipient puis valider la cellule 4

Voici les résultats obtenus dans le Notebook

```
# fermeture du port série
serial_port.close()
```

tensionU	tensionUther	Rther	temps	temperature
5.00	3.70	28461.54	0.00	1.35
5.00	3.59	25460.99	1.00	3.85
5.00	3.47	22679.74	2.00	6.50
5.00	3.37	20674.85	3.00	8.65
5.00	3.27	18901.73	4.00	10.77
5.00	3.20	17777.78	5.00	12.24
5.00	3.14	16881.72	6.00	13.49
5.00	3.08	16041.67	7.00	14.74
5.00	3.03	15380.71	8.00	15.77
5.00	2.99	14875.62	9.00	16.60
5.00	2.95	14390.24	10.00	17.42
5.00	2.93	14154.59	11.00	17.84
5.00	2.90	13809.52	12.00	18.46

Le graphique : Temperature = f(Temps)



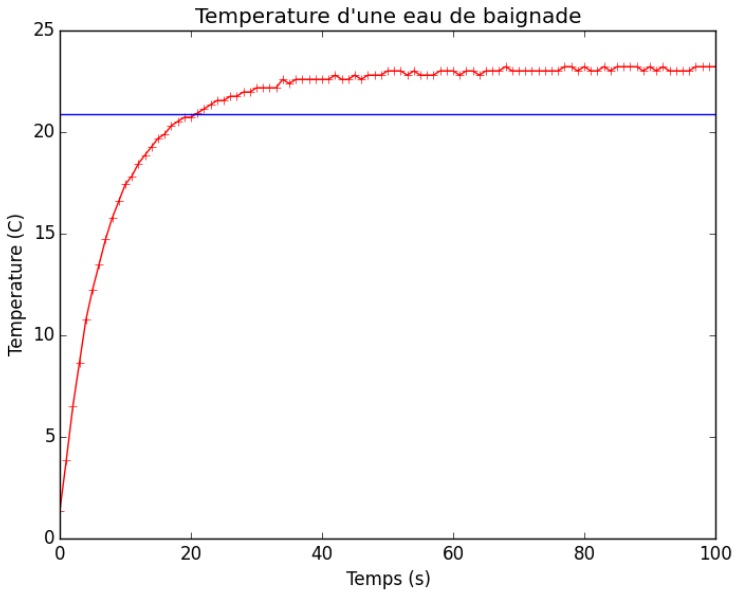
Exploitation

Le temps de réponse $t_{90\%}$ peut ainsi être calculé facilement avec Python

```
1 Tinitial = temperature[0]                    # première valeur dans la liste
2 Tfinal = np.mean(temperature[60:-1])         # une moyenne sur les dernières valeurs
3 T = Tinitial + 0.9 * (Tfinal - Tinitial)     # température à t90%
4 print('Tinitial = {0:.2f} C ; Tfinal = {1:.2f} C'.format(Tinitial, Tfinal))
5 print('T(t90%) = {0:.2f} C'.format(T))
```

```
Tinitial = 1.35°C ; Tfinal = 23.08°C
T(t90%) = 20.90°C
```

On peut ainsi facilement tracer la droite $Temperature = T$ avec l'instruction
`plt.axhline(y=T,color='b')`
Le réticule permet de lire la valeur de $t \simeq 20.7s$

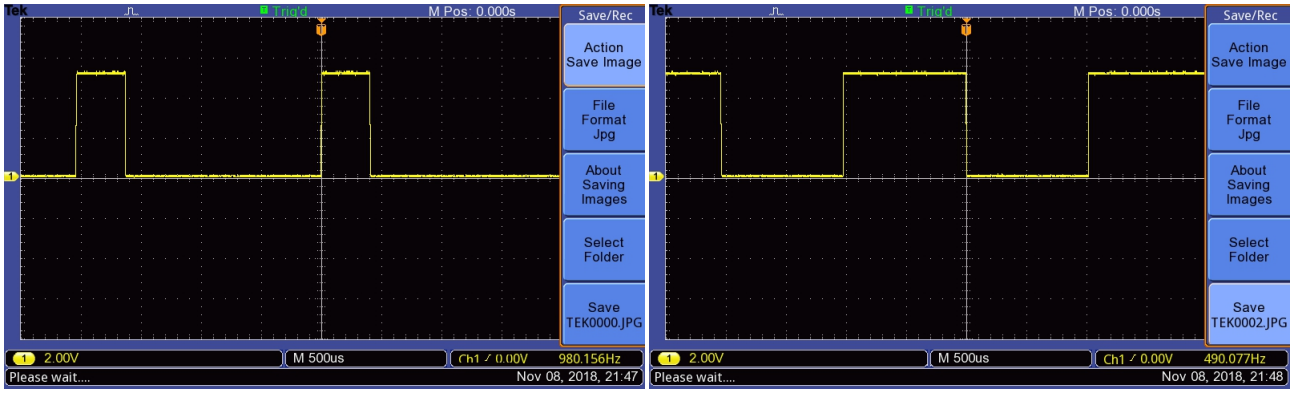


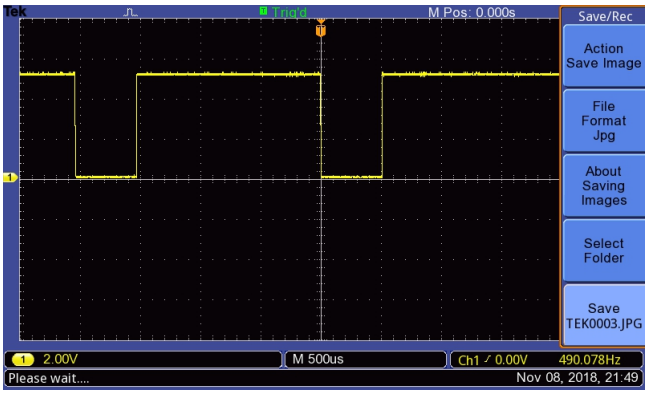
4.7 Modulation par largeur d'impulsion (MLI)

Les sorties numériques 3, 5, 6, 9, 10, 11 de la carte Arduino Uno (celles précédées d'un ~) sont capables de générer un signal modulé par largeur d'impulsion qui va osciller très rapidement entre 0V et 5V mais **sans jamais prendre de valeurs intermédiaires**. Attention la fréquence du signal n'est pas la même pour toutes les broches. Nous avons une fréquence d'environ 490 Hz pour les broches 3, 9, 10, 11 et une fréquence d'environ 980 Hz pour les broches 5 et 6. Nous pouvons visualiser le signal fourni par la sortie 11 à l'aide d'un oscilloscope grâce au code suivant en remplaçant *intensite* par une valeur comprise dans l'intervalle [0, 255] :

```
1  void setup(){
2    pinMode(11, OUTPUT);
3    analogWrite(11, intensite);
4  }
5
6  void loop(){
7  }
```

Pour les valeurs d'intensité respectives de 51, 128, 191 nous obtenons :



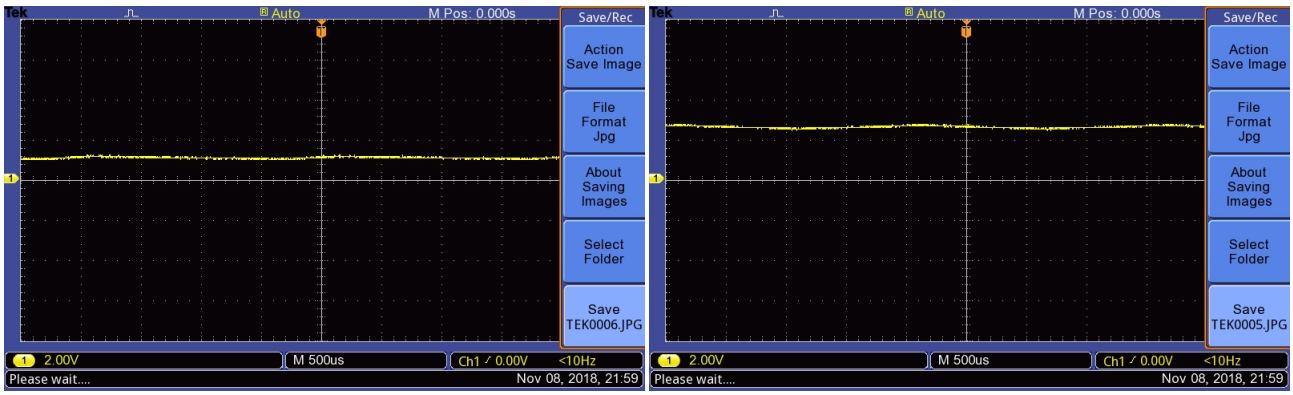


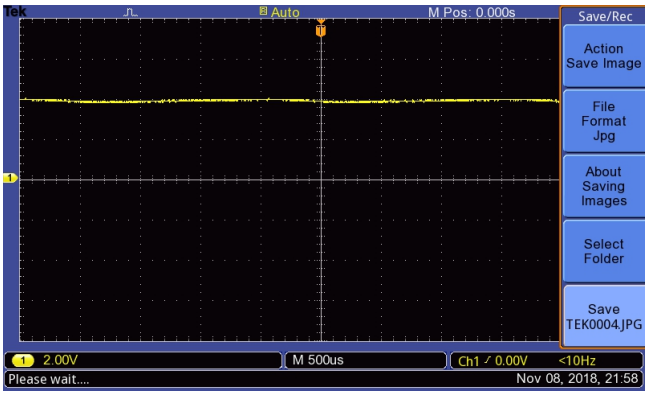
La valeur d'intensité représente le rapport cyclique du signal. S'il est à 0% le signal de sortie est toujours à 0 volt, de même lorsque le rapport cyclique est à 100% (donc une intensité de 255) le signal de sortie est toujours à 5 volts. Pour une valeur d'intensité de 51 le rapport cyclique est à 20% et le signal est égal à 5 volts pendant 20% du temps.

38. Quel est la valeur du rapport cyclique pour des intensités de 128 et 191 ?

Il s'agit maintenant d'utiliser une de ces sorties pour tracer la caractéristique d'un dipôle résistif. Pour ce faire nous avons besoin d'une tension analogique continue. L'idée est de filtrer le signal avec un filtre passe-bas pour ne conserver que la composante continue de notre signal. Il faut donc que le filtre possède une fréquence de coupure inférieure à la fréquence fondamentale du signal PWM.

Avec une résistance de 15 kΩ, un condensateur de 1 μF et des intensités de 51, 128 et 191 nous obtenons

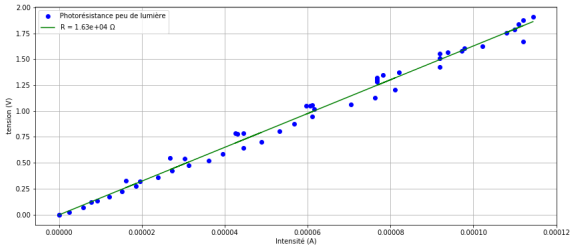


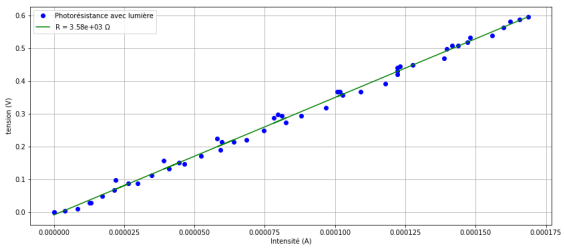


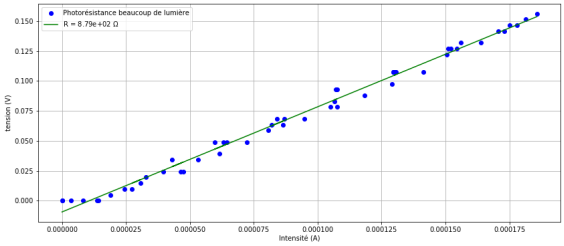
39. Ajouter au montage une résistance de 10 kΩ et une photorésistance pour tracer la caractéristique de cette dernière pour trois éclairages différents.

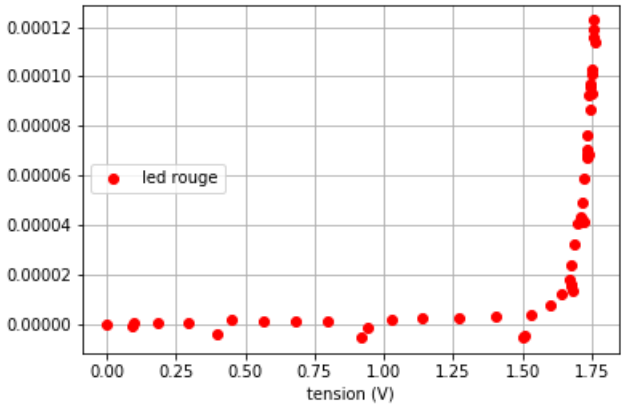
40. Modifier le code Arduino et concevoir le code Python pour obtenir le résultat suivant

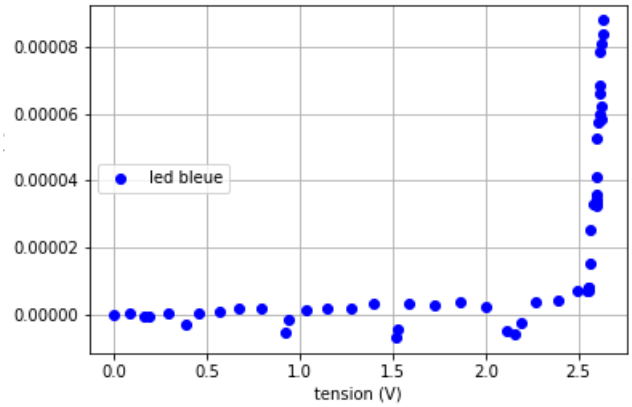
41. Faire de même avec une diode RGB pour tracer la caractéristique sur chaque couleur.

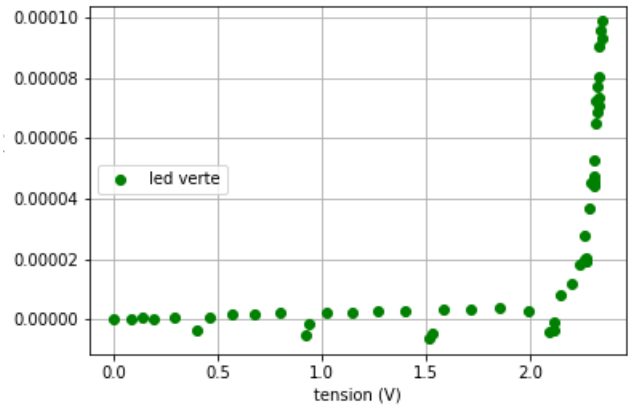







led rouge
tension (V)


led bleue
tension (V)


led verte
tension (V)

5 Lecture et sauvegarde des données avec Python

Dans un premier temps vous pouvez continuer votre lecture en passant directement au paragraphe 7.1. Il est possible de revenir un peu plus tard à cette partie plus technique sur le stockage des données.

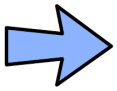
5.1 Utilisation d'un fichier CSV

Pour échanger, partager et analyser les données collectées lors d'une expérience, elles doivent être enregistrées dans un fichier au format informatique ouvert. Le plus utilisé est le format de fichier CSV (**C**omma-**S**eparated **V**alues). Ce type de format est en général proposé par tous les logiciels d'acquisition de données (Régressi, Latis Pro, Synchronie...) Lors de la sauvegarde les informations contenues dans votre tableur sont transformées en un fichier texte, par opposition aux formats dits binaires. Chaque ligne de texte de votre fichier correspond alors à une ligne de votre tableur et un **délimiteur** comme la virgule ou le point virgule correspondent aux séparations entre les colonnes. Les portions de texte séparées par une virgule correspondent ainsi aux contenus des cellules du tableau. La première ligne de ce fichier contient en général les titres de colonnes (grandeur mesurée en physique-chimie).

Exemple de fichier CSV

- Séparateur de colonnes : le point virgule.
- Le # indique un commentaire contenant du texte non convertible en valeur numérique.
- La virgule est utilisée comme séparateur de la partie entière et décimale.

	A	B	C
1	L	1/L	f0
2	20	0,05	161,4
3	25	0,04	127
4	30	0,033333333	105
5	35	0,028571429	89,7



Fichier CSV

```
# L; 1/L; f0
20; 0,05; 161,4
25; 0,04; 127
30; 0,033333333; 105
35; 0,028571429; 89,7
```

5.2 Lire les données contenues dans un fichier CSV

Pour lire un fichier, ici pas d'explorateur de fichiers, il faut indiquer à Python le répertoire dans lequel le fichier à lire se trouve. Il faut donc obtenir le répertoire par défaut dans lequel Python effectue la lecture ou l'enregistrement d'un fichier. Pour cela on utilise le package **os** qui permet de gérer le système d'exploitation. Puis on demande le répertoire courant de travail.

```
1 import os              #operating system
2 print(os.getcwd())     #c=current w=working d=directory
```

On peut ensuite modifier le répertoire courant de travail

```
1 os.chdir("C:\chemin\_absolu\_repertoire")  # Depuis de la racine
```

5.2.1 Le module : CSV

La modification est permanente, il n'est plus nécessaire d'indiquer le chemin du fichier à lire si celui-ci se trouve dans le répertoire de travail. Pour lire un fichier CSV avec Python on utilise le package **csv**. Il propose un objet `reader` permettant de décoder un fichier CSV. L'exemple le plus simple que l'on puisse écrire est le suivant :

```
1 import csv                                  # le module pour les fichiers csv
2 file = open("mon_fichier.csv", "r")         # ouvrir le fichier
3 reader = csv.reader(file, delimiter = ";") # initialisation d'un lecteur de fichier
4 for row in reader:                          # parcours du lecteur avec une boucle
5     print row                               # affichage ligne à ligne
6 file.close()                                # fermeture du fichier
```

Quelques précisions :

- ligne 2 : Le paramètre "r" de la fonction `open` impose l'ouverture du fichier en lecture seule. Dans ce cas il n'est pas possible de modifier son contenu.
- ligne 3 : Le délimiteur de colonnes est un point-virgule. Même s'il n'existe pas de spécification formelle pour l'écriture d'un fichier CSV, le séparateur par défaut est la virgule. Le point-virgule est surtout utilisé dans les pays, comme la France, où la partie décimale d'un nombre est précédée d'une virgule.

42. À l'aide d'un éditeur de texte ou d'un logiciel de gestion d'acquisition de données (LatisPro, Regressi, ...) élaborer un fichier CSV.
    Remarques :
    - Word ou Libre Office ne sont pas des éditeurs de texte, il faut utiliser Notepad sous Windows, gedit sous Linux ou TextEdit sous macOS. Attention de bien indiquer **l'extension csv** lors de la sauvegarde : mon_fichier.csv
    - Si vous utiliser un logiciel du type LatisPro, il y a normalement un menu permettant d'**exporter** vos données au format CSV.
43. Placer ensuite ce fichier dans le répertoire de travail définit avec Python, puis effectuer la lecture de ce fichier. Attention à bien indiquer le bon symbole pour le délimiteur de colonnes.

La lecture d'un fichier CSV réalisée sur LatisPro permet d'obtenir l'affichage suivant :

```
1 ['Longueur onde', 'Absorbance']
2 ['4E-7'; '0,3287']
3 ['4,05E-7'; '0,3546']
4 ['4,1E-7'; '0,3731']
```

L'objectif est maintenant l'exploitation des données récupérées du fichier CSV.

- On observe que chaque ligne du tableur de LatisPro est placée dans une liste Python. Or si ces données sont destinées à tracer des grahiques avec Python, nous venons de voir que les valeurs d'une même grandeur doivent appartenir à une même liste. Ce qui n'est manifestement pas le cas.
- On remarque également que toutes les valeurs sont considérées comme chaîne de caractères. Une conversion automatique des nombres est possible, mais elle ne fonctionne pas si la séparation partie entière, partie décimale est une virgule.

Il s'avère donc nécessaire d'écrire une fonction de lecture des fichiers CSV un peu moins naïve afin de tenir compte des remarques précédentes. Dans ce but, je propose ci-dessous la fonction `readColCSV` permettant d'extraire une colonne correspondant à la liste des valeurs de la grandeur désirée dans le fichier CSV. Cette fonction ne permettra pas de gérer tous les cas de figure que vous pourriez rencontrer lors de l'utilisation de fichier CSV mais c'est un bon point de départ que l'on peut ensuite enrichir en fonction de ses besoins.

Taper la fonction `readColCSV` dans une cellule du Notebook.

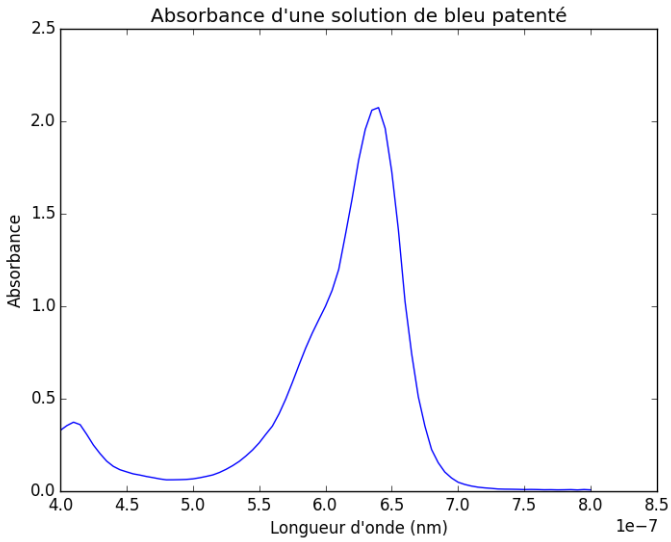
```
1  def readColCSV(fichier, sep, n):
2      '''
3      Pour les deux premiers paramètres attention à bien utiliser les guillements
4      car la fonction attend des chaines de caractères.
5      fichier <str> : Le nom du fichier -> "mon_fichier.csv"
6      sep     <str> : Le séparateur des colonnes par exemple -> ";"
7      n       <int> : Le numéro de la colonne à lire
8      '''
9      file = open(fichier , "r")
10     reader = csv.reader(file, delimiter = sep)
11     col = []
12     for row in reader:
13         try:
14             notation_point = row[n].replace(",", ".")
15             col.append(float(notation_point))
16         except:
17             pass
18     file.close()
19     return col
```

Pour utiliser cette fonction rien de plus simple, dans une cellule du Notebook, on tape

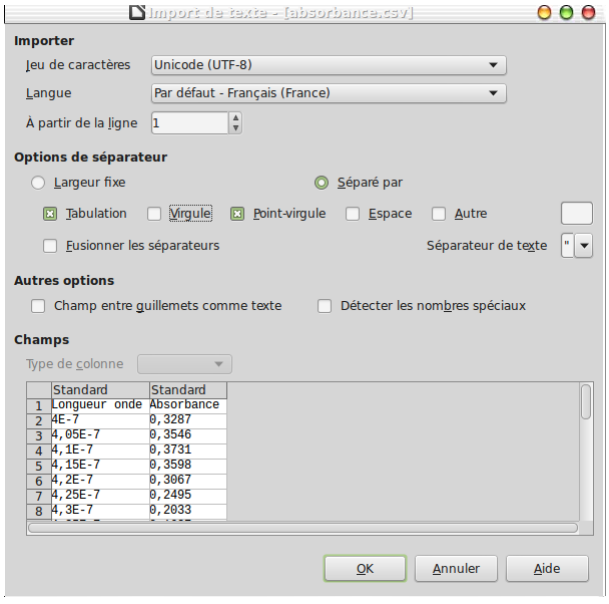
```
1 # On récupère les deux premières colonnes du fichier
2 x = readColCSV("absorbance.csv", ";", 0)
3 y = readColCSV("absorbance.csv", ";", 1)
```

Nous pouvons ensuite tracer le graphique correspondant à $y = f(x)$

```
1  # Affichage graphique
2  import matplotlib.pyplot as plt # Si vous ne l'avez pas déjà chargé
3  %matplotlib inline              # dans le même notebook
4
5  plt.plot(x,y)
6  plt.xlabel("Longueur d'onde (nm)")
7  plt.ylabel("Absorbance")
8  plt.title("Absorbance d'une solution de bleu patenté")
9  plt.savefig("absorbance.png")  # permet de sauvegarder votre graphique
10 plt.show()
```



Attention aux caractères accentués dans les chaînes de caractères car suivant la version Python, ils sont plus ou moins bien supportés. Pour ne pas avoir de problème avec les accents il est préférable d'utiliser une version > 3. Avec les versions antérieures il faut ajouter au début du code la ligne : `#-*- coding: utf-8 -*-` et faire précéder la chaîne de caractères d'un "u", par exemple pour le titre du graphique on écrira : `u"Absorbance d'une solution de bleu patenté"`



Pour lire les données d'un fichier CSV on peut également se servir d'un outils comme LibreOffice Calc. Au momment de l'ouverture du fichier le logiciel propose une fenêtre permettant de choisir entre autre :

- Le type d'encodage des caractères
- Le séparateur de colonne, attention aux virgules

Il en est de même lors d'un enregistrement vous pouvez choisir le format CSV et indiquer le séparateur.

5.2.2 Le module : pandas

Pour manipuler et analyser les données, le module panda, développé pour Python, s'avère des plus pratique. En particulier, il permet la lecture et l'écriture de fichiers csv de manière très simplifiée. Nous le chargeons donc dès le début de notre code.

Pour ouvrir un fichier en *lecture* et charger les données dans la mémoire de l'ordinateur, nous utilisons la fonction `read_csv` du module `panda`. Celle-ci reçoit plusieurs arguments parmi lesquels :

- le nom du fichier ;
- une indication sur le séparateur des données : virgule, point-virgule, espace, ou encore tabulation `\t` ;
- une option précisant si la première ligne du fichier est une ligne d'entêtes ou une ligne de données.

Si nous reprenons le fichier `absorbance.csv`, nous pouvons écrire :

```
1 # nom du fichier
2 filename = 'absorbance.csv'
3 # lecture des données et stockage dans la variable data
4 data = pd.read_csv(filename, sep = ';')
```

Pour avoir un aperçu du contenu (appelé dataframe), il ne reste plus qu'à écrire

```
1 data.head() # affiche les 5 premières lignes du fichier
```

```
   Longueur onde  Absorbance
0  4E-7           0,3287
1  4,05E-7        0,3546
2  4,1E-7         0,3731
3  4,15E-7        0,3598
4  4,2E-7         0,3067
```

Vous remarquerez que nous avons toujours le problème de la virgule comme séparateur de la partie entière et de la partie décimale, `panda` possède une fonction capable de remplacer d'un seul coup toutes les virgules en points.

```
1 # On remplace
2 data = data.replace(to_replace=',' , value=".", regex=True)
3 # On affiche
4 data.head()
```

```
   Longueur onde  Absorbance
0  4E-7           0.3287
1  4.05E-7        0.3546
2  4.1E-7         0.3731
3  4.15E-7        0.3598
4  4.2E-7         0.3067
```

Si la première ligne du dataframe contient les intitulés des colonnes, il est possible de récupérer les données de la colonne en y faisant référence. Ne pas oublier de convertir les données sous forme de valeurs numériques avec la fonction `astype('float')`

```
1 Lambda = data["Longueur onde"].astype('float')
2 Absorb = data["Absorbance"].astype('float')
```

44. Afficher le graphe de l'absorbance en fonction de la concentration

Dans le cas ou le séparateur décimal est un *point* (systèmes anglo-saxons) lors de l'ouverture d'un fichier de données, `pandas` transforme automatiquement les chaines de caractères en nombres entiers ou flottants suivant le contexte.

```
1 # nom du fichier
2 filename = 'systeme_anglo_saxon.csv'
3 # lecture des données et stockage dans la variable data
4 data = pd.read_csv(filename, sep = ';')
5 # affichage sous forme d'un tableau
6 data.head()
```

```
   x         y
0  0.312346  1.637363
1  0.535826  2.171534
2  0.430338  1.865677
3  0.283906  1.569671
4  0.365483  1.825927
```

```
1 print(data['x'].dtypes, data['y'].dtypes)
```

```
float64 float64
```

5.2.3 Quelques remarques utiles pour pandas

1. Si le fichier CSV ne possède pas d'en-tête de colonnes : on l'indique lors de la lecture du fichier avec le paramètre `hearder` et les colonnes sont affectées d'un indice entier. La première colonne possède l'indice 0, la deuxième colonne l'indice 1 et ainsi de suite.

   ```
   1 data = pd.read_csv(filename, sep = ';', header = None)
   ```

   Attention si rien n'est indiqué, ce sont les premières valeurs qui servent d'en-tête de colonnes
2. Les colonnes doivent me servir dans un calcul : il est possible d'ajouter une colonne à votre tableau (`dataframe`). Soit :

   ```
      A  B
   0  1  4
   1  2  5
   2  3  6
   ```

   ```
   1 data['C'] = data['A'] * data['B']
   2 data.head()
   ```

   ```
      A  B  C
   0  1  4  4
   1  2  5  10
   2  3  6  18
   ```

5.3 Enregistrer les données de l'acquisition dans un fichier CSV

5.3.1 Avec le module : CSV

Maintenant que nous savons lire un fichier CSV et même extraire une colonne de ce fichier, l'écriture n'est guère plus compliquée. Le module *csv* définit un `reader` pour la lecture et bien il définit de même un `writer` pour l'écriture. Le programme minimum permettant d'écrire dans un fichier est le suivant :

```
1 file_name = "out.csv"
2 file = open(file_name, "w")
3 writer = csv.writer(file)      # Création de l'écrivain CSV.
4
5 writer.writerow( ("x", "y") )  # Écriture de la ligne d'en-tête des colonnes
6 writer.writerow( (1.80, 3.6) ) # Écriture de quelques données
7 file.close()
```

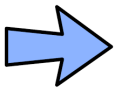
Il ne reste plus qu'à écrire une fonction capable de sauvegarder les données obtenues lors de la mesure de la fréquence du stroboscope. Pour cela il faut :

- Créer un fichier CSV
- Parcourir les listes obtenues, `intensite` et `temps`, valeur par valeur

- Ajouter la valeur pour un même indice de chaque liste comme une nouvelle ligne de notre fichier CSV.

Les listes Python

```
1 temps = [1, 1.5, 2]
2 intensite = [100, 50, 100]
```



Le fichier CSV

temps , intensité
1 , 100
1.5 , 50
2 , 100

La fonction `writeCSV` accepte en arguments deux chaînes de caractères pour le nom de fichier et le séparateur ainsi que deux listes python. Cette fonction d'écriture très simple permet de satisfaire les situations rencontrées lors de cette formation.

```
1  def writeCSV(fichier, sep, col1, col2):
2      '''
3      fichier <str>  : Le nom du fichier CSV à créér -> "mon_fichier.csv"
4      sep     <str>  : Le séparateur des colonnes
5      col1    <list> : La première colonne -> [1, 1.5, 2, ...]
6      col2    <list> : La deuxième colonne -> [100, 50, 100, ...]
7      '''
8      file = open (fichier , "w" )
9      writer = csv.writer(file, delimiter=sep)
10     fin1, fin2 = len(col1), len(col2)
11     if fin1 == fin2:
12         for i in range(fin1):
13             writer.writerow( (col1[i], col2[i]) )
14     else:
15         print("Les deux listes n'ont pas la même taille")
16     file.close()
```

3. Exécuter la fonction avec les listes `temps` et `mesure` de l'activité 4.4.
4. Ajouter deux arguments à la fonction permettant d'écrire la ligne d'en-tête (noms de colonnes) du fichier CSV.

5.3.2 Avec le module : pandas

Si nous reprenons l'exemple précédent avec les listes python `temps` et `intensite` nous pouvons écrire le code suivant permettant de sauvegarder les données dans un fichier CSV.

```
1 data = pd.DataFrame({
2     'Temps'     : temps,
3     'Intensité' : intensite
4 })
5 filename = "mesures_intensite.csv"
6 data.to_csv(filename, sep=';', index=False, encoding='utf-8')
```

- `sep=';'` indique que les informations sont séparées par un point-virgule;
- `index=False` précise qu'aucun indice de ligne doit être enregistré dans le fichier;
- `encoding='utf-8'` stipule que l'encodage des données dans le fichier est *utf-8*

5.3.3 Stockage des données : CLIMAT

De nombreux capteurs permettent d'étudier l'évolution du climat de notre palnète. Ces dernières années des moyens considérables ont été déployés pour observer, mesurer, modéliser et simuler les facteurs influençant le climat de notre planète. Prenons comme exemple les intéractions océan-atmosphère, plusieurs outils sont utilisés pour mesurer la température, la pression et la salinité des océans en surface et en profondeur. Les mesures sont réalisées à l'aide

de sondes ou de bouées puis sont transmises grâce à une balise Argos. Pour plus de sûreté, les mesures sont aussi enregistrées sur une carte mémoire interne à la sonde ou à la bouée.

Une campagne de mesures liée à la température des océans a été réalisée à l'aide d'une sonde dérivante capable d'effectuer des cycles programmés de descente jusqu'à 2000 m de profondeur. Les caractéristiques de cette campagne sont les suivantes :

- durée de la campagne : 1 mois
- fréquence d'échantillonnage : 0.1 Hz

Les relevés de la campagne de mesure sont écrits dans un fichier texte dont le contenu est défini comme suit.
Les informations relatives à la campagne se trouvent sur les deux premières lignes du fichier, on y trouve, la date, le numéro de la sonde, le numéro de la campagne, etc.
Pour les lignes suivantes sur chaque ligne on trouve la température en kelvins indiquée par 5 caractères, 1 caractère séparateur et 4 caractères pour la profondeur en mètre ainsi qu'un caractère de fin de ligne. Voici quelques lignes en exemple :

```
293.5,0005
...
289.7,0100
...
277.8,1500
...
```

5. On suppose que chaque caractère est codé sur 8 bits, en ne tenant pas compte des deux premières lignes, déterminer le nombre d'octets enregistrés en une heure.
6. En déduire le nombre approximatif d'octets contenus dans le fichier de cette campagne. Une carte de 1 Go est-elle suffisante ?

6 Utilisation d'une carte SD avec Arduino UNO

6.1 Rendre la carte Arduino autonome.

La carte arduino peut très bien être utilisée sans qu'elle soit branchée à l'ordinateur. Dans ce cas il est nécessaire d'alimenter la carte par l'intermédiaire du port d'alimentation ou bien à l'aide de la borne d'entrée Vin. Les deux ont les mêmes caractéristiques, les tensions d'alimentation recommandées sont comprises entre 7V et 12V (en savoir plus).

Une fois la carte autonome, il est possible de faire de l'acquisition avec une carte arduino mobile. Nous avons alors un véritable **système embarqué**. C'est à dire un système autonome capable de réaliser une tâche précise en temps réel.

Dans ce cadre il devient primordial de pouvoir sauvegarder l'information acquise à l'aide des capteurs branchés sur la carte. Pour cela nous allons utiliser un **shield SD** (caractéristiques).

6.2 Écrire des données sur la carte SD.

La documentation pour la SD Library. Vous y trouverez tous les renseignements nécessaires sur la communication entre le microcontroleur et la carte SD à travers le bus SPI.

6.2.1 Initialiser la carte SD

```
#include <SPI.h> // Pour la communication SPI
#include <SD.h>  // Pour la communication avec la carte SD

void setup() {
  /* Initialisation du port série */
  Serial.begin(9600);
  /* Initialisation de la carte SD */
  SD.begin(10);
}
void loop(){
  // Votre code
}
```

6.2.2 Écriture sur la carte

Pour notre premier exemple, nous allons enregistrer les changements de valeur d'un compteur dans un fichier nommé `data.csv`
Avant cela nous commencerons par créer le fichier à l'aide la fonction open

DESCRIPTION DE LA FONCTION open

Opens a file on the SD card. If the file is opened for writing, it will be created if it doesn't already exist (but the directory containing it must already exist).

Syntax
SD.open(filepath)
SD.open(filepath, mode)

Parameters
filename : the name the file to open, which can include directories (delimited by forward slashes,) - char *
mode (optional) : the mode in which to open the file, defaults to FILE_READ - byte. one of :
FILE_READ : open the file for reading, starting at the beginning of the file.
FILE_WRITE : open the file for reading and writing, starting at the end of the file.

Returns
a File object referring to the opened file; if the file couldn't be opened, this object will evaluate to false in a boolean context, i.e. you can test the return value with "if (f)".

Dans un souci de simplicité, le choix a été fait d'effacer le fichier `data.csv` s'il existe déjà afin de pas l'allonger à chaque nouvelle acquisition.

```
#include <SPI.h>
#include <SD.h>

/* Nom du fichier d'enregistrement */
File fichier;
/* Le compteur */
int i = 0;

void setup() {

  Serial.begin(9600);
  SD.begin(10);

  if (SD.exists("data.csv"))                // Si le fichier existe sur la carte
    SD.remove("data.csv");                  // Il est effacé
  fichier = SD.open("data.csv", FILE_WRITE); // Création d'un nouveau fichier
  fichier.println("Compteur");               // Information à écrire sur la carte
  fichier.flush();                          // On force l'écriture
}

void loop() {
  fichier.println(i);
  i = i + 1;
  fichier.flush();
}
```

7. Réaliser une acquisition avec un capteur en conservant les données sur la carte SD. Penser à formater vos données au format CSV.
8. Que fait le code suivant?

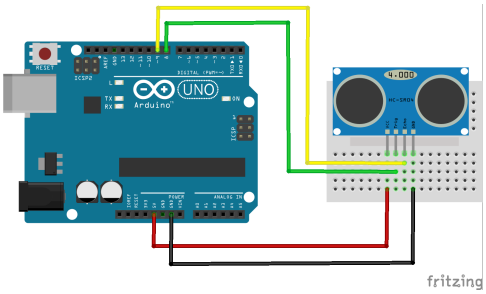
```
int numero = 0;
while (SD.exists("data" + String(numero) + ".csv"))
  numero = numero + 1;
fichier = SD.open("data"+ String(numero)+ ".csv", FILE_WRITE);
```

9. Comment l'insérer dans votre code pour obtenir l'effet escompté?

7 À vous de jouer

7.1 Mesure de la vitesse du son

7.1.1 Le capteur ultrason



7.1.2 La réalisation

Les objectifs

Réaliser le traitement d'un flux de données provenant d'un capteur ultrason afin de déterminer la vitesse du son. Le contrôle de l'acquisition est effectué avec une carte *Arduino UNO*, le traitement et la modélisation sont réalisés avec le langage de programmation Python. Les étapes essentielles de ce projet sont :

- Écrire le programme Arduino pour l'acquisition des données.
- Récupérer avec Python les mesures de temps pour des distances comprises entre 10 cm et 50 cm.
- Tracer avec Python le graphique de la distance en fonction du temps.
- Sauvegarder les données (temps, distance) dans un fichier CSV.

Document nº 1 : **Le code Arduino et Python**

```
1  // Déclaration des variables globales : broches
2
3  // Broche TRIGGER
4  // Broche ECHO
5
6  void setup() {
7
8    pinMode(trigg, OUTPUT);       // Configuration des broches
9    digitalWrite(trigg, LOW);     // La broche TRIGGER doit être à LOW au repos
10   pinMode(echo, INPUT);         // La broche ECHO en entrée
11
12   // À compléter                // Démarrage de la liaison série
13 }
14
15 void loop() {
16
17   digitalWrite(trigg, HIGH);    // Lance une mesure de distance en envoyant
18   delayMicroseconds(10);        // une impulsion HIGH de 10 microsecondes
19   digitalWrite(trigg, LOW);
20
21   temps = pulseIn(echo, HIGH); // Mesure le temps en microseconde entre
22                                // l'envoi de l'ultrason et sa réception
23
24   // À compléter                // Les résultats sur le port série
25                                // On fait une pause
26 }
```

Un petit bout de programme Python pour tester la fonction `input` et donner des idées sur la manière de gérer le flux de données envoyé par la carte Arduino.

```
1 mesure = float(input("Entrez votre mesure : "))
2 while mesure != -1:
3     serial_port.flushInput()
4     print(serial_port.readline())
5     mesure = float(input("Entrez votre mesure : "))
```

Document n° 2 : La fonction `input()`

Il est souvent utile que l'utilisateur puisse entrer des données au clavier. La méthode la plus simple consiste à employer la fonction native de Python : `input()`. Cette fonction provoque une interruption dans le programme courant. L'utilisateur est invité à entrer des caractères au clavier et à valider en appuyant sur <Enter>. Lorsque cette touche est enfoncée, l'exécution du programme se poursuit, et la fonction fournit en retour une chaîne de caractères correspondant à ce que l'utilisateur a entré. Cette chaîne peut alors être référencée par un nom de variable quelconque.
On peut invoquer la fonction `input()` en laissant les parenthèses vides. On peut aussi y placer en argument un message destiné à l'utilisateur.
Attention le nom de cette fonction dépend de votre version de Python

- Python 2.7 : `raw_input()`
- Python 3.x : `input()`

Document n° 3 : Caractéristiques techniques du module HC-SR04

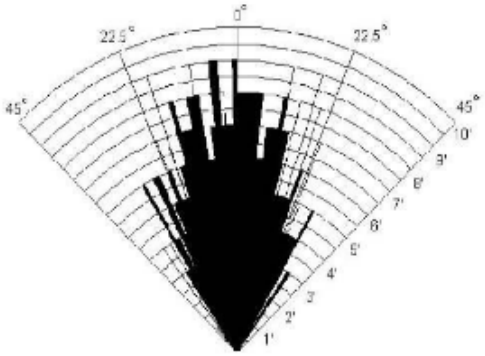
Le capteur HC-SR04 utilise les ultrasons pour déterminer la distance d'un objet. Il offre une excellente plage de détection sans contact, avec des mesures de haute précision et stables. Son fonctionnement n'est pas influencé par la lumière du soleil ou des matériaux sombres, bien que des matériaux comme les vêtements puissent être difficiles à détecter.

Caractéristiques

- Dimensions : 45 mm x 20 mm x 15 mm
- Plage de mesure : 2 cm à 400 cm
- Résolution de la mesure : 0.3 cm
- Angle de mesure efficace : 15^o
- Largeur d'impulsion sur l'entrée de déclenchement : 10 μs (Trigger Input Pulse width)

Broches de connection

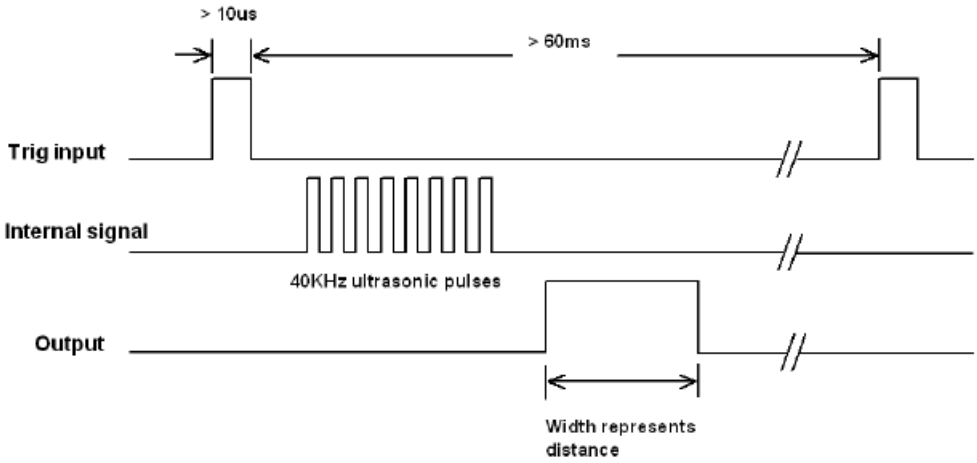
- Vcc = Alimentation +5 V DC
- Trig = Entrée de déclenchement de la mesure (Trigger input)
- Echo = Sortie de mesure donnée en écho (Echo output)
- GND = Masse de l'alimentation



Practical test of performance,
Best in 30 degree angle

Document nº 4 : Caractéristiques techniques du module HC-SR04

Pour déclencher une mesure, il faut une impulsion "high" (5V) d'au moins 10 μs sur l'entrée "Trig". Le capteur émet alors une série de 8 impulsions ultrasoniques à 40 kHz, puis il attend le signal réfléchi. Lorsque celui-ci est détecté, il envoie un signal "high" sur la sortie "Echo", dont la durée est proportionnelle à la distance mesurée.

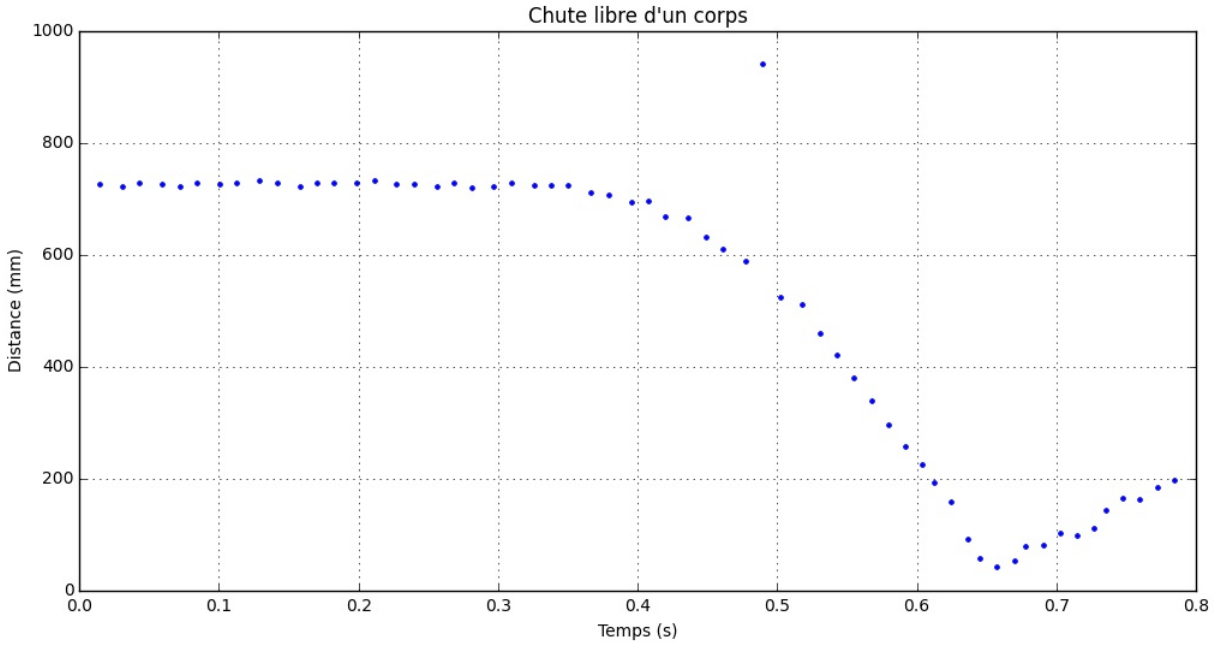


7.2 Des idées pour la suite...

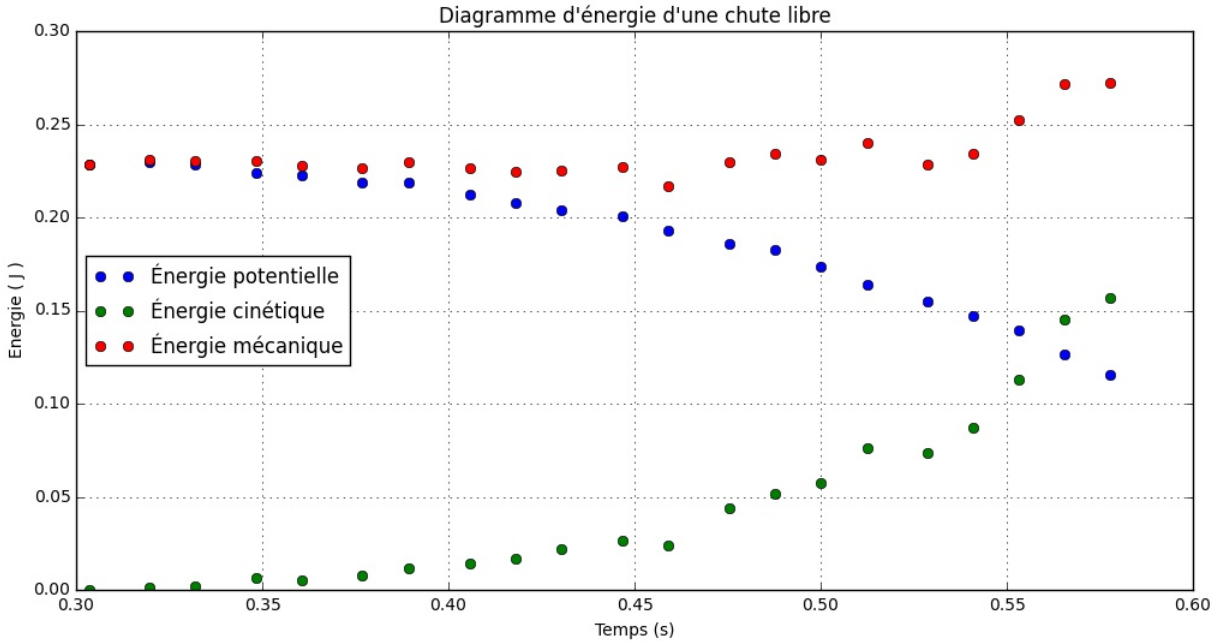
Afin de préparer la suite de la formation, je vous donne quelques idées de montage à réaliser avec la carte Arduino UNO en lien avec le programme de TS. Pour chaque montage je vous donne la représentation graphique des résultats que j'ai obtenus

7.2.1 Chute libre d'un corps sans vitesse initiale.

Avec le matériel dont vous disposez essayer d'imaginer un montage et son programme permettant d'obtenir la courbe représentative de la loi horaire de la chute libre d'un corps.

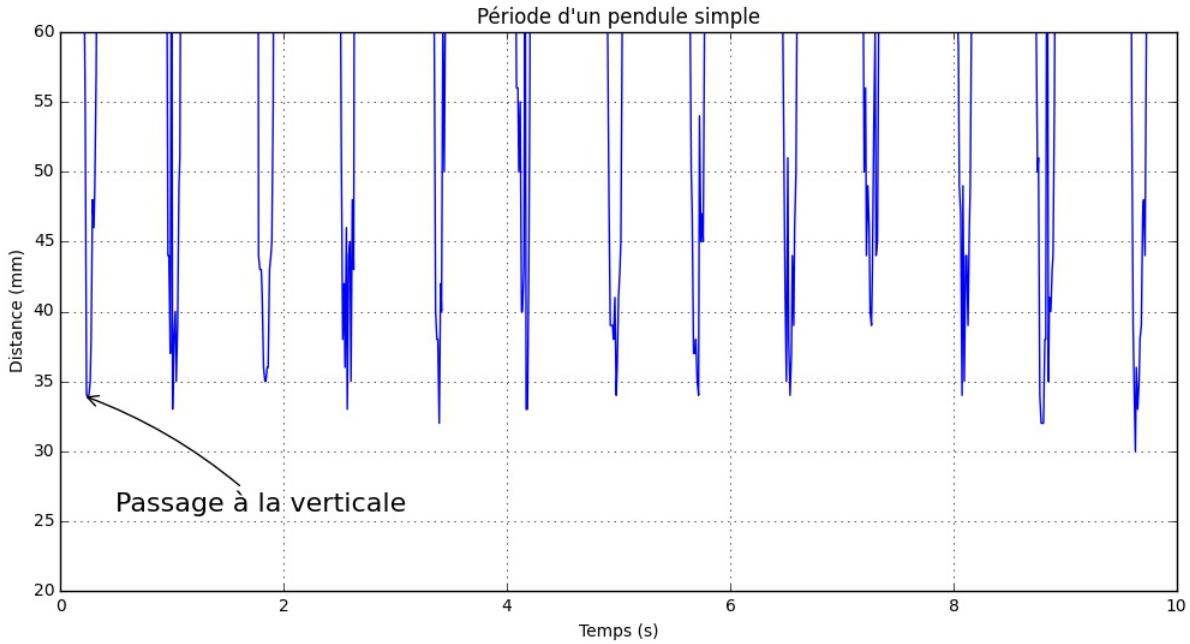


On pourra également tracer les courbes d'énergie cinétique, potentielle et mécanique.



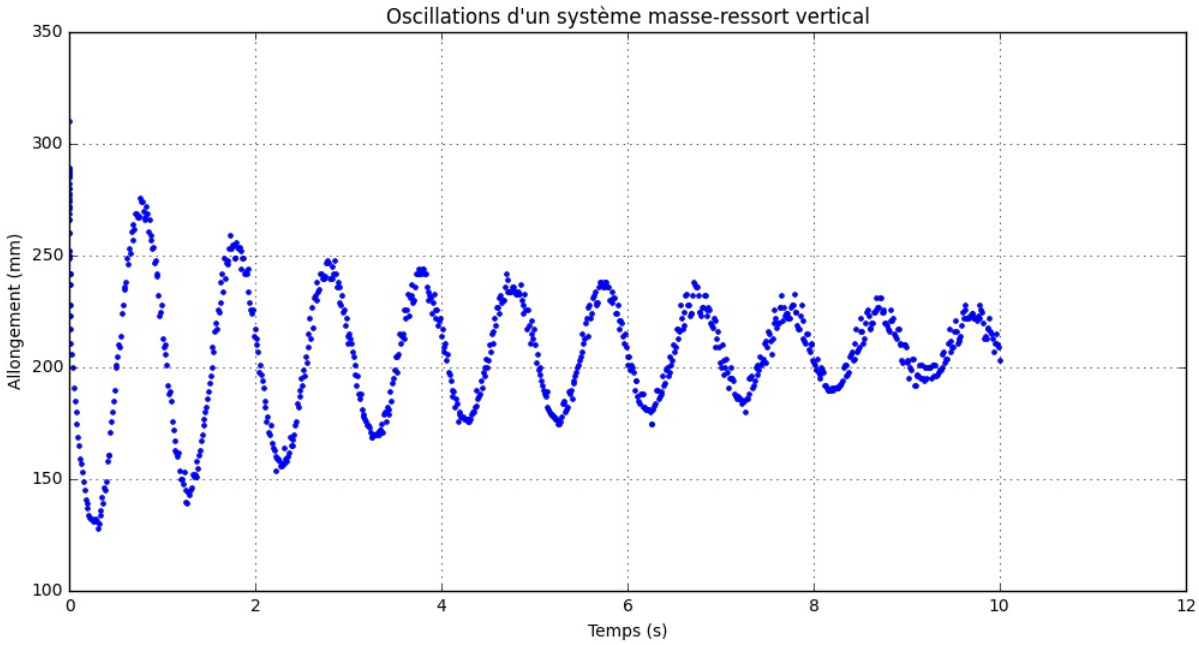
7.2.2 Période d'un pendule simple.

Avec le matériel dont vous disposez essayer d'imaginer un montage et son programme permettant de mettre en évidence la période d'un pendule simple.



7.2.3 Oscillateur : système solide-ressort vertical.

Avec le matériel dont vous disposez essayer d'imaginer un montage et son programme permettant d'obtenir l'évolution temporelle de l'allogement d'un ressort.


Oscillations d'un système masse-ressort vertical
Allongement (mm)
Temps (s)
350
300
250
200
150
100
0
2
4
6
8
10
12

8 Introduction au traitement du signal.

8.1 Un signal c'est quoi ?

Dans le cadre de cette formation nous utiliserons la définition d'un signal comme la représentation physique d'une information à transmettre ou de façon équivalente comme une entité qui sert a véhiculer une information.

Attention toute grandeur physique qui varie au cours du temps n'est pas un signal. En effet les signaux sont tout le temps « bruité ». Le « Bruit » est l'ensemble des phénomènes perturbateurs qui gênent voire empêchent la perception ou l'interprétation de l'information suivant le rapport Signal sur Bruit. La grandeur physique véhiculant le signal doit être mesurable et pourvue d'une unité. Les grandeurs électriques sont souvent utilisées comme signaux transportant l'information. Mais on trouve aussi fréquemment des signaux lumineux, sonores ou radio.

Cette définition du signal amène à associer la grandeur physique véhiculant le signal à sa description mathématique sous la forme d'une fonction du temps.

8.2 Continuité et discontinuité en temps

Un signal peut-être représenté par une fonction qui varie dans le temps notée $f : t \mapsto y(t)$, on peut distinguer plusieurs cas possibles suivant les valeurs prises par la variable t :

- Les **signaux à temps continus**, les valeurs $f(t)$ sont définies pour $t \in \mathbb{R}^+$. Par convention, on suppose que la variable temps ne peut pas prendre de valeurs négatives.
- Les **signaux à temps discret**, les valeurs $f(t)$ n'existent que pour des valeurs discrètes de t, lors d'une acquisition les valeurs de t sont sont prises à intervalle constant.

On distingue également les **signaux non quantifiés** pour lesquels f peut prendre n'importe quelle valeur dans un intervalle continu et les **signaux quantifiés** pour lesquels f ne peut prendre que des valeurs discrètes. Dans le premier cas on parle parfois de signaux analogiques et dans le deuxième cas on parle parfois de signaux numériques. Attention de ne pas généraliser, les signaux analogiques et numériques sont des sous ensembles respectifs des signaux continus et discrets.
Remarques : Si les signaux continus peuvent être modélisés par des fonctions mathématiques plus ou moins complexes il n'en est pas de même avec les signaux discrets. Il est plus facile de les définir à l'aide d'une suite de valeurs notée $(y_i)_{i\in\mathbb{N}}$ où les y_i représentent les $f(t_i)$. Cette suite de valeurs est aussi souvent appelée **échantillons** en physique-chimie.
Une autre différence importante à faire pour le traitement du signal est celle qui permet de distinguer les *signaux déterministes* des *signaux stochastiques*. Pour le premier on peut grâce à une étude mathématique prévoir les valeurs qui vont être prises par le signal dans le temps alors que pour le second la valeur à l'instant t_i ne nous renseigne pas sur la valeur à un instant t_{i+1}

8.2.1 Dérivée discrète

D'après un document de Jean-Luc Charles, enseignant chercheur à l'ENSAM

Reprenons les résultats expérimentaux obtenus lors de la chute libre d'un point matériel M sans vitesse initiale. Nous avons fait l'acquisition des positions d'un point matériel de masse m en fonction du temps. Ces valeurs permettent également de tracer le graphique d'évolution des énergies (cinétique, potentielle et mécanique) lors d'une chute libre en calculant, en autres, la vitesse instannée en différents points de la chute.

La suite de ce document porte sur la difficulté d'obtenir un calcul acceptable pour le valeur de la vitesse instantannée. Ce qui revient à s'interroger sur le calcul de la dérivée d'une suite discrète de valeurs au cours du temps.

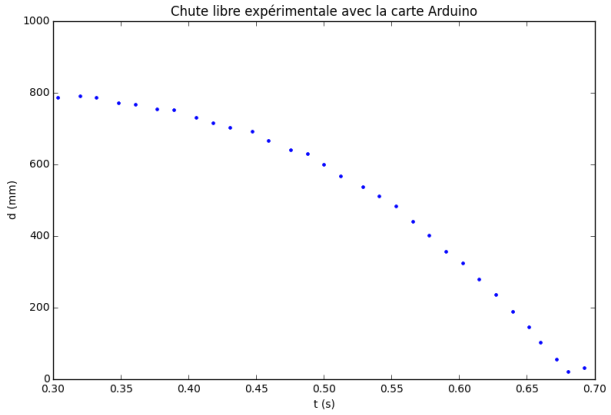
Soit un point matériel M de masse m en chute libre.
On note $(z_i)_{i\in\mathbb{N}}$ la suite de valeurs permettant de repérer l'altitude du point M au cours du temps. z_i est l'altitude du point M à un instant $t_i \geq 0$.
On note $(z'_i)_{i\in\mathbb{N}}$ la suite des valeurs de la vitesse du point matériel M au cours du temps. z'_i est la vitesse de M à un instant $t_i \geq 0$.
Le calcul de la vitesse du point M à un instant $t_i \geq 0$, encore appelé dans ce cas **dérivée discrète** à l'instant $t_i \geq 0$, peut se définir de différentes manières :

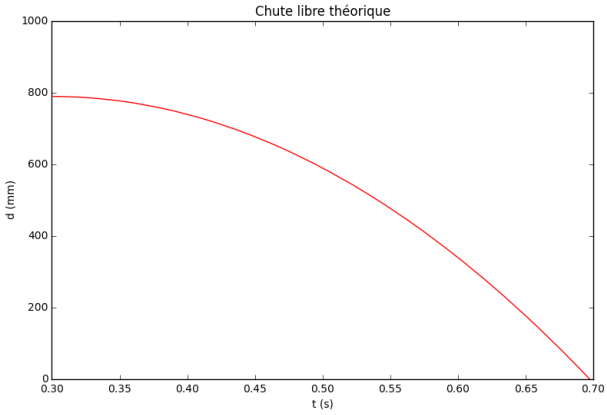
- Dérivée à gauche : $z'_i = \dfrac{z_i - z_{i-1}}{t_i - t_{i-1}}$

- Dérivée centrée : $z'_i = \dfrac{z_{i+1} - z_{i-1}}{t_{i+1} - t_{i-1}}$
- Dérivée à droite : $z'_i = \dfrac{z_{i+1} - z_i}{t_{i+1} - t_i}$

La loi horaire de la chute libre est donnée par : $z(t) = h - \dfrac{1}{2} g t^2$

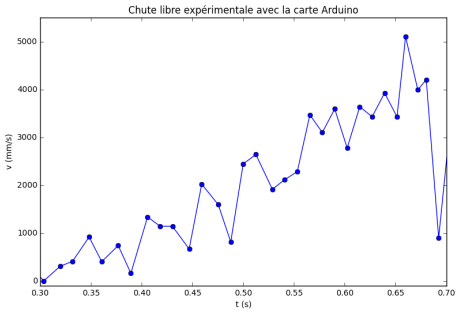
Les graphiques de la loi horaire de la chute libre, expérimentale et théorique, sont les suivants :

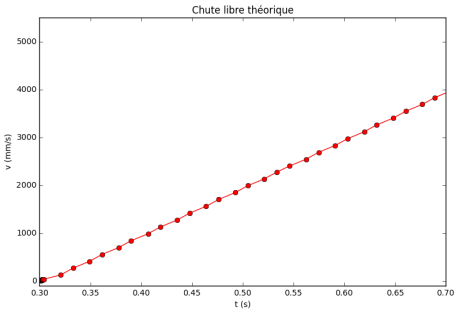




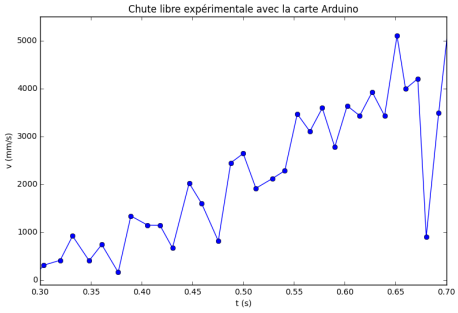
Maintenant nous allons calculer les vitesses du point matériel M en utilisant les trois relations (gauche, droite et centrée) avec les positions expérimentales et théoriques.

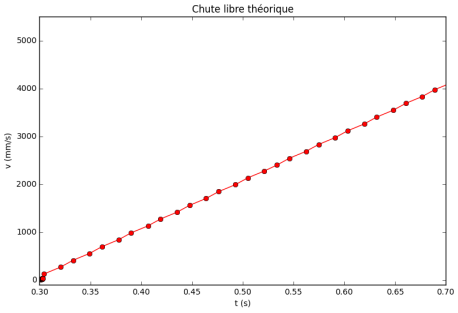
Gauche



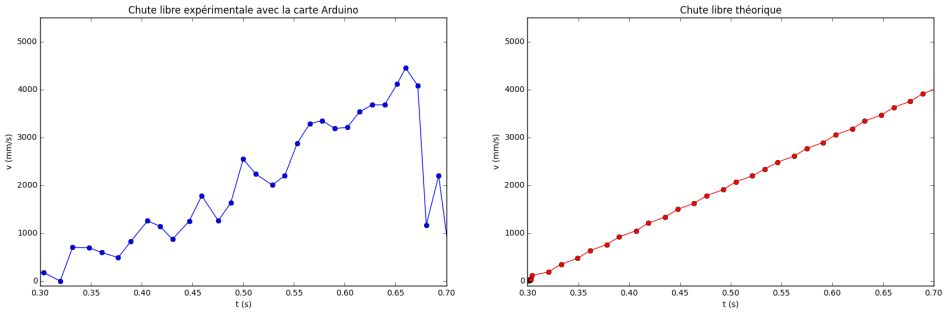


Droite





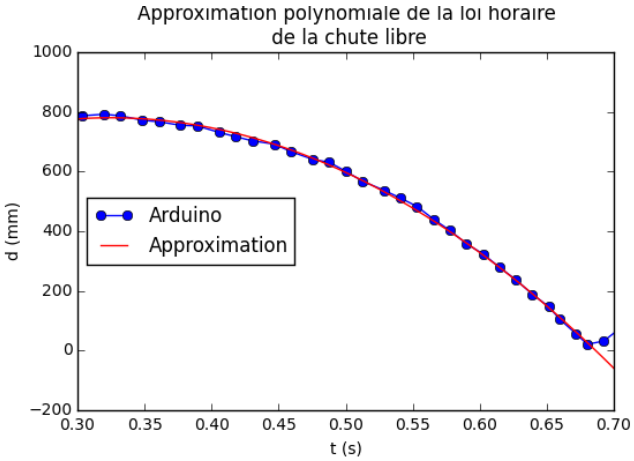
Centrée



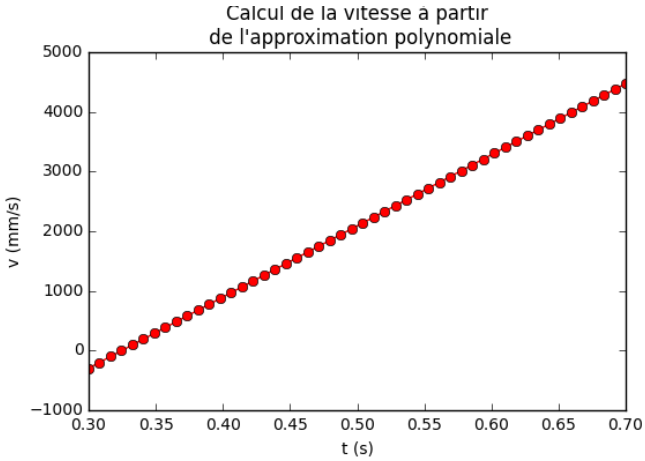
On observe que les résultats obtenus pour les calculs de la vitesse instantannée ne sont pas satisfaisants.

8.2.2 Calcul de la dérivée par approximation polynomiale

L'idée ici est de faire une approximation polynomiale de la suite $(z_i)_{i\in\mathbb{N}}$ par une fonction $Z : t \mapsto Z(t)$ sur l'intervalle de temps $[t_0, t_0 + T]$ avec $T \in \mathbb{R}^+$. t_0 représente le temps initial et on cherche une approximation polynomiale du mouvement pour $t \in [t_0, t_0 + T]$. La dérivée de cette fonction notée Z' s'obtient alors par simple dérivation du polynôme obtenu.



Approximation polynomiale $Z : t \mapsto -5961t^2+3871t+151.7$



Représentation graphique de $|Z'|$

Pour calculer l'approximation polynomiale des points de la suite $(z_i)_{i\in\mathbb{N}}$ on utilise la fonction `polyfit` du module `numpy`. La fonction `polyfit` retourne les coefficients du polynôme représentant la fonction Z, qui peuvent ensuite être utilisés avec numpy.poly1d. Puis on trace la vitesse en dérivant la fonction Z obtenue.

Pour cela on utilise les fonction `polyfit` et `poly1d` du module numpy. La première renvoie l'ensemble des coefficients du polynome, de degré 2 dans l'exemple, ajusté par la méthode des moindres carrés, la deuxième permet de construire la fonction mathématique et Python correspondantes. Dans l'exemple suivant je n'ai conservé que les points intéressants de l'acquisition c'est à dire ceux de l'intervalle : [45 : 75] des listes x et y.

```
1 import numpy as np
2 z = np.polyfit(x[45:75], y[45:75], 2)
3 p = np.poly1d(z)
```

8.2.3 Moyenne et énergie d'un signal

À COMPLÉTER

9 ANNEXES

9.1 Arduino RGB

```
/* Broches */
const int LED_R = 8;
const int LED_G = 9;
const int LED_B = 10;

void setup(){
  // Initialise les broches
  pinMode(LED_R, OUTPUT);
  pinMode(LED_G, OUTPUT);
  pinMode(LED_B, OUTPUT);
}

void loop(){
  // 1 seconde par couleur
  digitalWrite(LED_R, HIGH);
  delay(1000);
  digitalWrite(LED_R, LOW);
  digitalWrite(LED_G, HIGH);
  delay(1000);
  digitalWrite(LED_G, LOW);
  digitalWrite(LED_B, HIGH);
  delay(1000);
  digitalWrite(LED_B, LOW);
}
```

9.2 Arduino JCMB

```
/* Broches */
const int LED_R = 8;
const int LED_G = 9;
const int LED_B = 10;

void setup(){
  // Initialise les broches
  pinMode(LED_R, OUTPUT);
  pinMode(LED_G, OUTPUT);
  pinMode(LED_B, OUTPUT);
}

void loop(){
  digitalWrite(LED_R, HIGH);
  digitalWrite(LED_G, HIGH);
  delay(1000);
  digitalWrite(LED_R, LOW);
  digitalWrite(LED_B, HIGH);
  delay(1000);
  digitalWrite(LED_G, LOW);
  digitalWrite(LED_R, HIGH);
  delay(1000);
  digitalWrite(LED_G, HIGH);
  delay(1000);
  digitalWrite(LED_R, LOW);
  digitalWrite(LED_G, LOW);
  digitalWrite(LED_B, LOW);}
```

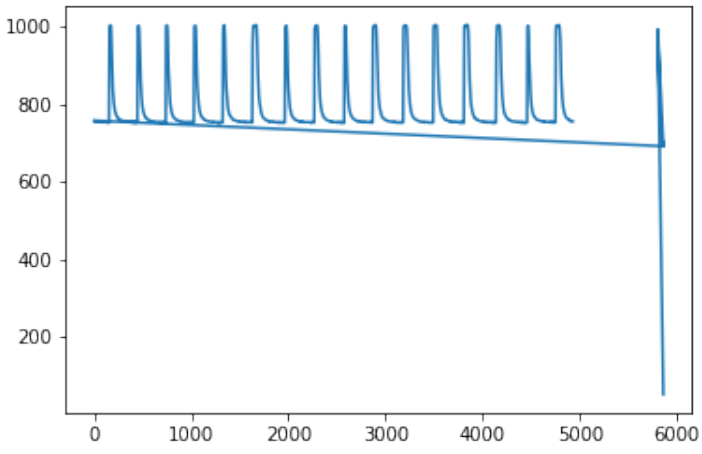
9.3 Arduino intensité d'un canal

```
/* Broches */
const int LED_R = 8;
const int LED_G = 9;
const int LED_B = 10;

void setup(){
  // Initialise les broches
  pinMode(LED_R, OUTPUT);
  pinMode(LED_G, OUTPUT);
  pinMode(LED_B, OUTPUT);
}

void loop(){
  analogWrite(LED_G, 0);
  delay(1000);
  analogWrite(LED_G, 75);
  delay(1000);
  analogWrite(LED_G, 125);
  delay(1000);
  analogWrite(LED_G, 250);
  delay(1000);
}
```

9.4 La photorésistance

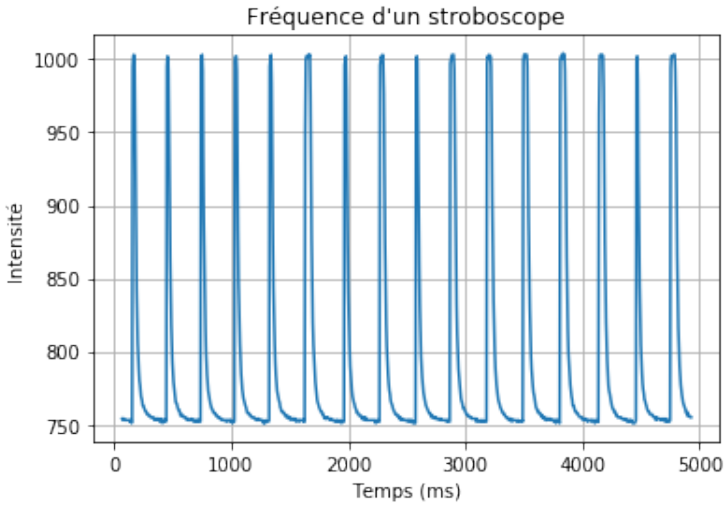
Le code arduino

```
const int broche = A0;

void setup() {
  Serial.begin(19200);
}

void loop() {
  int valeur = analogRead(broche);
  Serial.print(valeur);
  Serial.print("\t");
  Serial.println(millis());
}
```



On ne conserve que les valeurs exploitables. On rappelle qu'il est possible de n'utiliser qu'une partie des valeurs d'une liste, soit L une liste alors on écrit : `L[debut, fin]`. Dans notre exemple on peut essayer de démarrer à partir de la valeur d'index 50 en gardant toutes les valeurs suivantes, ce qui donne : `temps[50: ]`. À vous d'ajuster en fonction de ce que vous obtenez.

```
1 # attention les deux listes doivent contenir le même nombre de valeurs.
2 plt.plot(temps[50:], mesure[50:])
3
4 plt.title("Fréquence d'un stroboscope")
5 plt.ylabel('Intensité')
6 plt.xlabel('Temps (ms)')
7 plt.grid()
8 plt.show()
```



9.4.1 Le bouton poussoir

```
1  // Etat en cours de l'automate
2  int etat;
3  // Etat à mémoriser
4  int oldEtat;
5
6  //Les états possibles de l'automate
7  const int WAIT = 2;
8  const int START = 1;
9  const int STOP  = 0;
10
11 // Les broches utilisées
12 //capteur
13 const int broche = A0;
14 //bouton poussoir
15 const int BP = 3;
16
17 void setup() {
18   //initialisation des variables
19   oldEtat = LOW;
20   etat = WAIT;
21   //config E/S
22   pinMode(BP, INPUT);
23   //liaison série
24   Serial.begin(19200);
25 }
26
27 void loop() {
28   //Lecture du bouton
29   int etatBP = digitalRead(BP);
30   //gestion des états
31   if(oldEtat == LOW && etatBP == HIGH){
32     if (etat == WAIT)
33     {
34       etat = START;
35     }
36     else if (etat == STOP)
37     {
38       etat = START;
39     }
40     else if (etat == START)
41     {
42       etat = STOP;
43     }
44   }
45
46   //Traitement des états
47   if(etat == START){
48     int valeur = analogRead(broche);
49     Serial.print(valeur);
50     Serial.print("\t");
51     Serial.println(millis());
52   }
53   else if(etat == STOP)
54     Serial.println("-1\t -1");
55
56   oldEtat = etatBP;
57   delay(10);
58 }
```

9.5 Le projet : vitesse du son

9.5.1 Le code Arduino

```
// Déclaration des variables globales : broches
int trigg = 8;
int echo = 9;

void setup() {
  pinMode(trigg, OUTPUT);          // Configuration des broches
  digitalWrite(trigg, LOW);        // La broche TRIGGER doit être à LOW au repos
  pinMode(echo, INPUT);            // La broche ECHO en entrée
  Serial.begin(9600);              // Démarrage de la liaison série
}

void loop() {

  digitalWrite(trigg, HIGH);       // Lance une mesure de distance en envoyant
  delayMicroseconds(10);           // Une impulsion HIGH de 10 microsecondes
  digitalWrite(trigg, LOW);        // Fin d'émission
  int temps = pulseIn(echo, HIGH); // Mesure temps émission-reception

  Serial.print(temps);
  Serial.print("\t");              // on ajoute une tabulation et la
  Serial.println("-1");            // la valeur -1
  delay(500);
}
```

Dans cet exemple seule la valeur temps nous intéresse. Mais pour uniformiser le code nous ajoutons une tabulation et la valeur -1. Cela permet de réutiliser les codes Python précédents. En effet coté Python on attend toujours deux informations (deux grandeurs physiques). C'est un choix de programmation que nous avons fait dès le départ mais qui est cohérent avec les besoins que nous avons en physique-chimie qui se traduit souvent par l'étude d'une grandeur en fonction du temps. Il suffira d'ignorer la lecture de la valeur -1 coté Python. En informatique il est courant d'utiliser la valeur -1 pour indiquer soit une erreur soit une valeur à ignorer.

9.5.2 Le code Python

Penser à écrire une fonction pour calculer le temps moyen sur un nombre entier n de valeurs renvoyées par le capteur ultrason.

```
def temps_moyen(n, serial_port):
    """
    Cette fonction calcule une moyenne des temps ultrasons
    reçus sur n valeurs

    n           -> <int>    : Nombre de valeurs à lire
    serial_port -> <serial> : Port série
    """
    i = 0
    t_somme = 0
    serial_port.flushInput()
    while i < n:
        val = serial_port.readline().split()
        try:
            t = float(val[0])       # lecture temps ultrason
            t_somme += t            # la somme des temps ultrasons
            i = i + 1               # ajoute 1 à la variable comptant les mesures
        except:
            pass
    return t_somme/n
```

```
# les imports
import serial
import matplotlib.pyplot as plt
```

```
# ouverture du port série et synchronisation des données entre arduino et Python.
serial_port = serial.Serial( port = "/dev/ttyACM0", baudrate =9600)

temps = []      # une liste pour les mesures de temps
distances = []  # une liste pour les mesures de distance
dist = 0        # juste pour entrer dans la boucle
nb_t = 10       # le nombre de mesure temps

while dist != -1:
    dist = float(input("Entrez votre mesure : "))
    if dist != -1:
        distances.append(dist)
        tm = temps_moyen(nb_t, serial_port)
        temps.append(tm)

# fermeture du port série
serial_port.close()
```

9.5.3 Comment faire les mesures ?

- Téléverser le programme Arduino dans la mémoire de la carte.
- Ouvrir un moniteur série et vérifier que les mesures de temps s'affichent. Faire varier la distance entre l'objet et le capteur HC-SR04 pour observer que les valeurs temps augmentent si la distance augmente.
- Valider les cellules contenant le code Python concernant cet exercice.
- Placer correctement votre capteur HC-SR04 à une distance déterminée de votre objet.
- Normalement vous devez avoir la possibilité de saisir cette distance dans une boite de dialogue qui c'est ouverte sur le Notebook.

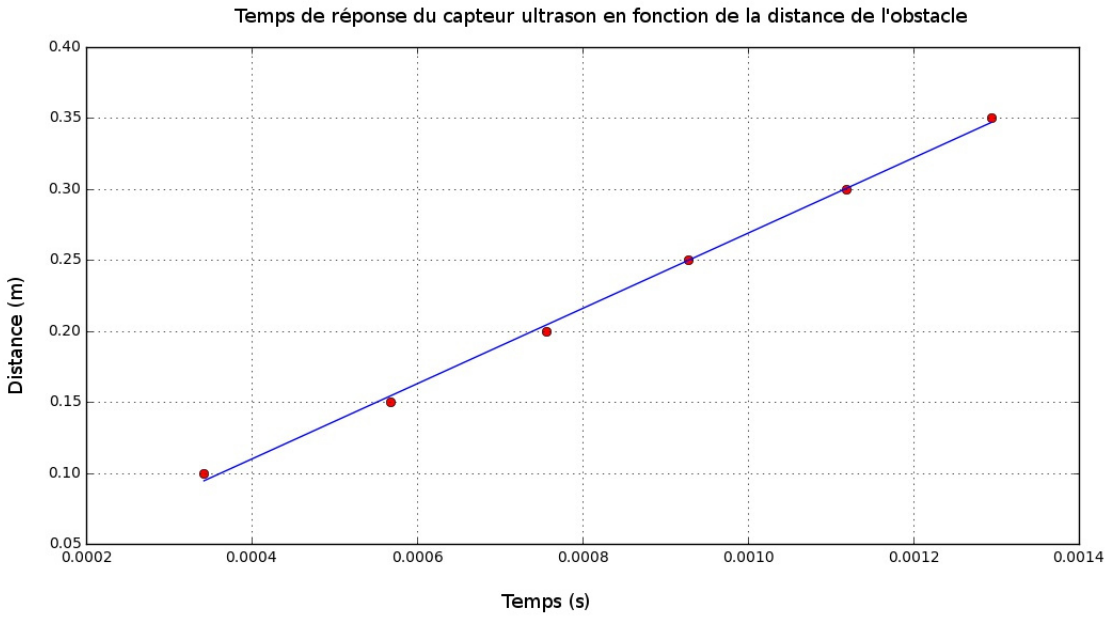
```
# fermeture du port série
serial_port.close()
```

```
Entrez votre mesure : 0.1
Entrez votre mesure : 0.15
Entrez votre mesure : 0.20
Entrez votre mesure : 0.25

Entrez votre mesure :
```

- Valider avec la touche *Enter*, modifier la distance capteur-objet, entrer la valeur, etc...

L'affichage du graphique : distances en fonction du temps, je vous laisse ajouter les commentaires : title, xlabel, ylabel

```
plt.figure(figsize=(12,6))
plt.plot(mesures, distances, 'ro')
plt.grid()
plt.show()
```



9.6 La chute libre

9.6.1 Le code Arduino

```
// Déclaration des variables globales : broches
int trigg = 8;
int echo = 9;


void setup() {

  pinMode(trigg, OUTPUT);           // Configuration des broches
  digitalWrite(trigg, LOW);         // La broche TRIGGER doit être à LOW au repos
  pinMode(echo, INPUT);             // La broche ECHO en entrée

  Serial.begin(19200);              // Démarrage de la liaison série
}

void loop() {

  digitalWrite(trigg, HIGH);        // Lance une mesure de distance en envoyant
  delayMicroseconds(10);            // Une impulsion HIGH de 10 microsecondes
  digitalWrite(trigg, LOW);         // Fin d'émission

  int temps_echo = pulseIn(echo, HIGH); // Mesure temps émission-reception

  Serial.print(temps_echo);
  Serial.print("\t");
  Serial.println(millis());
  delay(1);
}
```

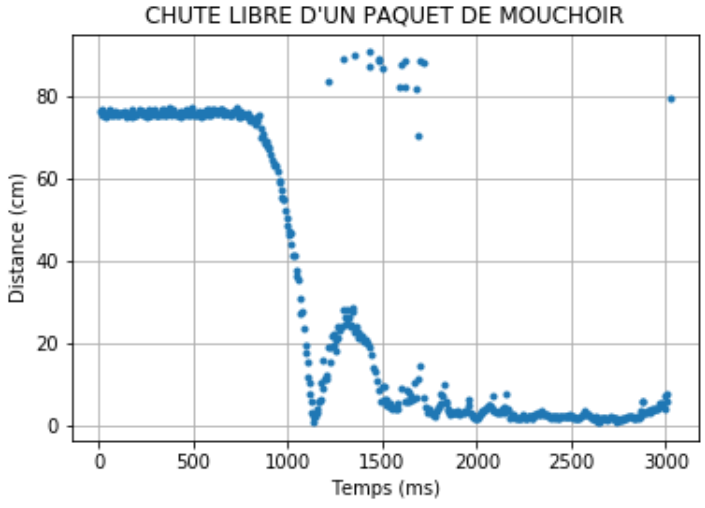
9.6.2 Le code Python

Les mesures ont été faites avec un paquet de mouchoirs en papier accroché à une ficelle d'environ 80 cm. J'ai lancé le code python puis j'ai compté jusqu'à deux avant de lacher le paquet bien à la verticale du capteur ultrason.

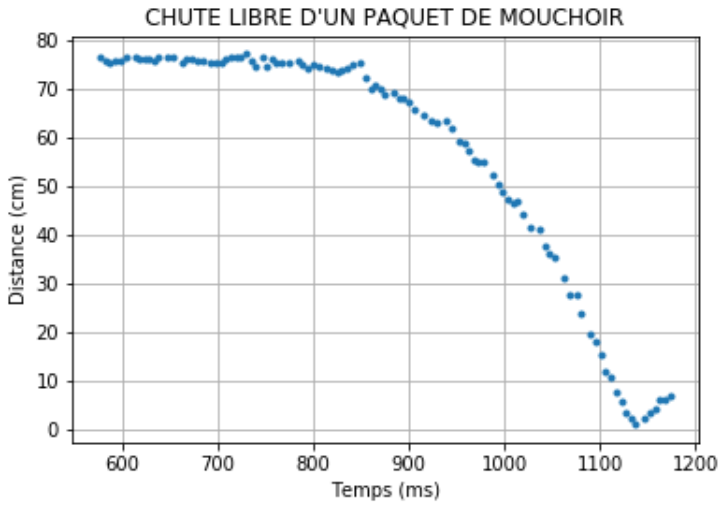
```python
# les imports
import serial
import time
import matplotlib.pyplot as plt
```

```python
# ouverture du port série
serial_port = serial.Serial( port = "/dev/ttyACM1", baudrate =19200)
serial_port.setDTR(False)
time.sleep(0.1)
serial_port.setDTR(True)
serial_port.flushInput()

# les mesures
mesure_dist = []
temps = []
duree = 3000                                    # durée d'acquisition
end = False

while end == False or temps[-1] - temps[0] <= duree:
    val = serial_port.readline().split()        # lecture des données
    try:
        d = float(val[0])/58                    # distance en cm
        t = float(val[1])                       # temps écoulé en ms
        if d > 1 and d < 100:                   # filtrage des valeurs aberrantes
            mesure_dist.append(d)               # entre 1cm et 1m
            temps.append(t)
            end = True
    except:
        pass
# fermeture du port série
serial_port.close()
```

```python
plt.plot(temps, mesure_dist, ".")
plt.title("CHUTE LIBRE D'UN PAQUET DE MOUCHOIR")
plt.ylabel('Distance (cm)')
plt.xlabel('Temps (ms)')
plt.grid()
plt.show()
```

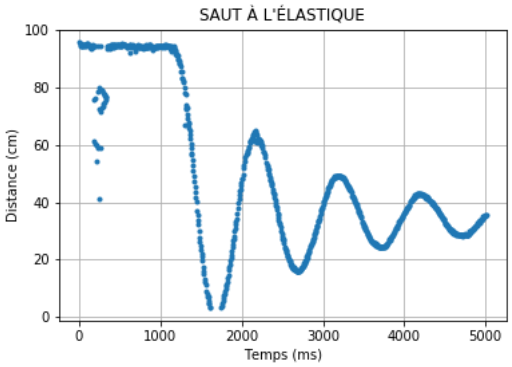


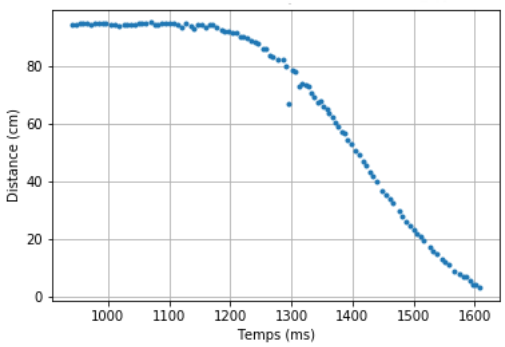
```
plt.plot(temps[100:200], mesure_dist[100:200], ".")
plt.title("CHUTE LIBRE D'UN PAQUET DE MOUCHOIR")
plt.ylabel('Distance (cm)')
plt.xlabel('Temps (ms)')
plt.grid()
plt.show()
```



9.7 Le saut à l'élastique

Avec un capteur ultrason il est possible il également possible d'étudier les différentes phases d'un saut à l'élastique.





Les codes Arduino et Python pour obtenir ces résultats sont exactement les mêmes que pour la chute libre. Il peut donc être intéressant de créer une fonction Python que les élèves pourraient utiliser à partir d'un module.

```
# ouverture du port série
serial_port = serial.Serial( port = "/dev/ttyACM0", baudrate =19200)
serial_port.setDTR(False)
time.sleep(0.1)
serial_port.setDTR(True)
serial_port.flushInput()

temps, mesure_dist = distance_ultrason(5000, serial_port)

# fermeture du port série
serial_port.close()
```

```python
def distance_ultrason(duree, serial_port, inf = 1, sup = 100):
    """
    Renvoie respectivement une liste temps (dates d'acquisition)
    et une liste distance en (cm) mesurée par le capteur ultrason
    durant une durée donnée et dans un intervalle de distance
    fixée par défaut entre 1cm et 1m

    duree       -> <float>  : durée de l'acquisition en (ms)
    serial_port -> <serial> : port série ouvert à la communication
    inf         -> <float>  : distance minimum d'acquisition en (cm)
    sup         -> <float>  : distance maximum d'acquisition en (cm)
    """
    mesure = []
    temps = []
    end = False
    while end == False or temps[-1] - temps[0] <= duree:
        val = serial_port.readline().split()    # lecture des données
        try:
            d = float(val[1])/58                # distance en cm
            t = float(val[0])                   # temps écoulé en ms
            if d > inf and d < sup:             # filtrage des valeurs aberrantes
                mesure.append(d)
                temps.append(t)
                end = True
        except:
            pass
    return temps, mesure
```