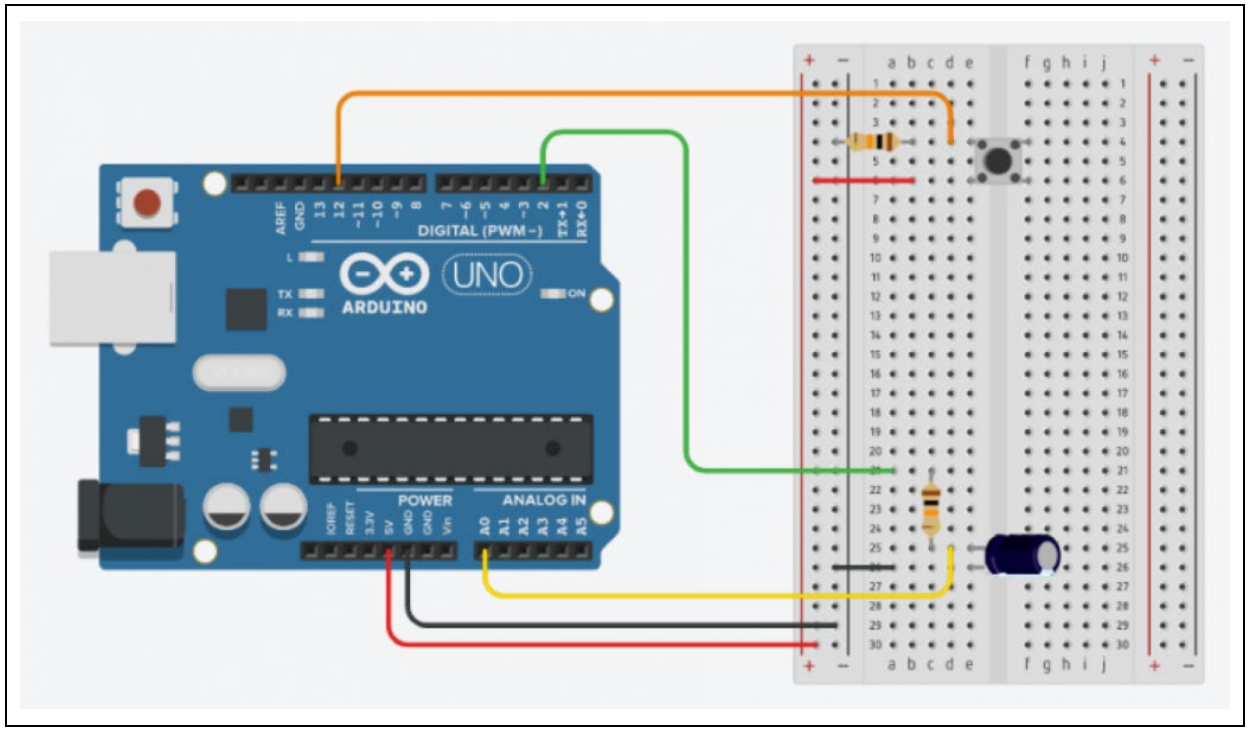
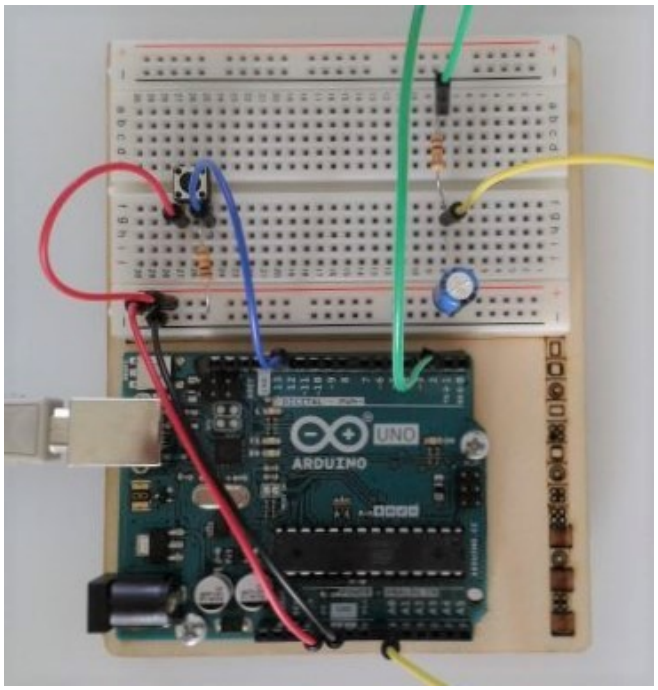
Projet 7 – Dipôles RC : Charge & Décharge

Avec un Arduino, nous avons vu qu'il était possible de mesurer une tension. Nous allons maintenant utiliser un Arduino Uno pour suivre l'évolution temporelle de la tension aux bornes d'un condensateur lors de sa charge ou sa décharge.

Toutes les activités seront réalisées avec le circuit suivant :



- Liste des composants :



.1 condensateur de 100 µF (C chimique : attention à la polarité)

. 2 résistances de 10 kΩ (résistance du bouton poussoir et du dipôle RC)

. 1 bouton poussoir

. 1 plaque d'essais

. Fils de connexion

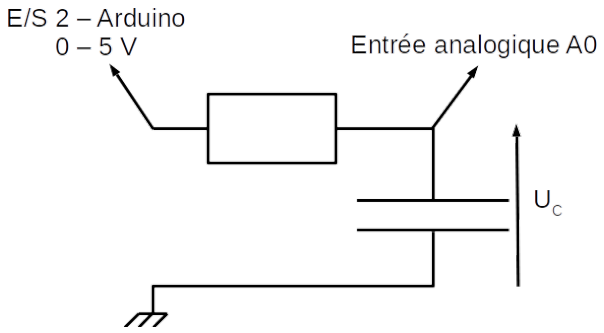
- Protocole de communication:

. Firmata Express

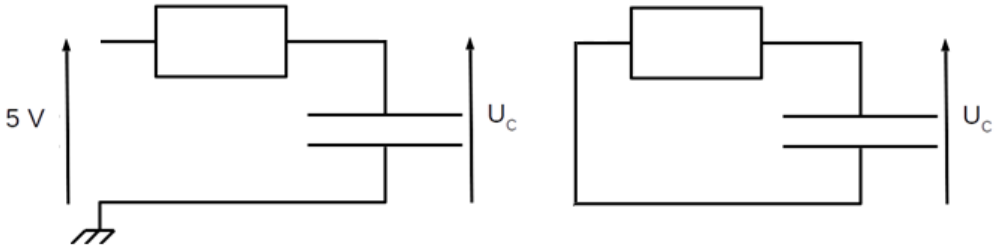
Dans ce montage, La carte Arduino est utilisée :

. pour appliquer une tension de 5 volts aux bornes du dipôle résistance - condensateur grâce à une sortie numérique (broche 2),

. pour mesurer la tension aux bornes du condensateur à l'aide d'une entrée analogique (broche A0).



Suivant l'état logique de broche N°2 déclarée en sortie numérique, le schéma électrique sera équivalent à :



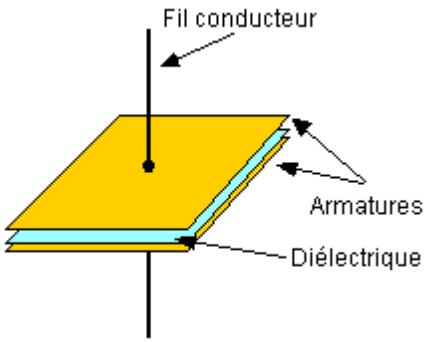
Broche N°2 à niveau haut **Broche N°2 à niveau bas**

Avec un Arduino, pour obtenir des mesures suffisamment précises, il faut utiliser un dipôle RC conduisant à des constantes de temps longues (quelques centaines de millisecondes au minimum).

Rappels :

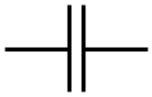
. Les condensateurs

Un condensateur est un composant électronique capable de stocker de l'énergie sous la forme d'un champ électrostatique. Il est constitué de 2 armatures séparées par un isolant, le "diélectrique".

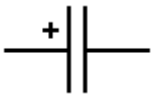


Les dimensions des armatures (ou plaques) sont grandes devant l'épaisseur de l'isolant. Par commodité, les films métalliques et l'isolant sont enroulés ou juxtaposés en plusieurs couches.

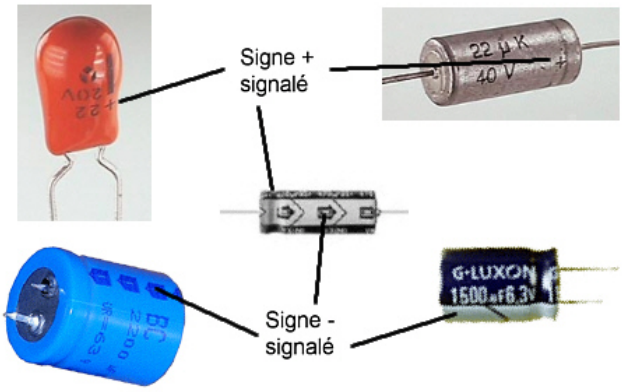
Dans les schémas de circuits électriques, le condensateur est symbolisé par :



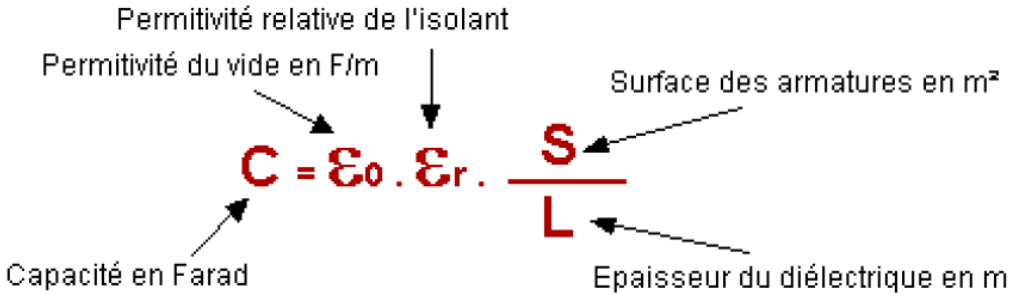
Les condensateurs avec un diélectrique chimique sont polarisés.
Ils sont représentés par ce symbole :



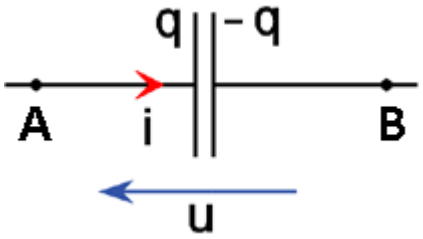
Les condensateurs polarisés possèdent un pôle "plus" et un pôle "moins", ils doivent impérativement être connectés dans le bon sens. En règle générale, les condensateurs polarisés radiaux (qui ont les deux pattes du même côté) possèdent une bande ou un ensemble de flèches qui désigne le pôle négatif, et les condensateurs polarisés axiaux (qui ont les deux pattes opposées) possèdent un renfoncement (collerette) côté pôle positif :



La caractéristique principale d'un condensateur est sa capacité exprimée en **farad.**



En appliquant, une tension U aux bornes d'un condensateur, il apparait une charge q, sur une de ses plaques, proportionnelle à la tension appliquée.



La charge qui apparaît sur l'une des plaques est l'opposé de celle qui apparaît sur l'autre plaque

En effet, la conservation de la charge électrique s'écrit : **q + q' = 0** d'où : **q' = – q**.

La charge q est, par convention la charge portée par la plaque par laquelle on entre dans le condensateur. On a alors :

q = C.U

où : q est exprimé en coulomb (C), U en volt (V) et C en Farad (F)

<u>Remarque</u> : Le farad étant une grande unité, on utilise souvent des sous-multiples :

1 µF = 10^{-6} F, 1 nF = 10^{-9} F, 1 pF = 10^{-12} F

L'intensité du courant, i, est le débit de charges et est égale à la quantité de charges qui passent par unité de temps à travers une section de conducteur :

$$i = \frac{dq}{dt}$$

On oriente géométriquement le condensateur de la borne A vers la borne B.

Soit i_{AB} l'intensité algébrique du courant allant de A vers B et qA la charge qui apparaît sur la plaque par laquelle on entre dans le condensateur.

On a donc : **qA = – qB = C.(VA – VB)**

En tenant compte de la relation entre i_{AB} et qA, on a :

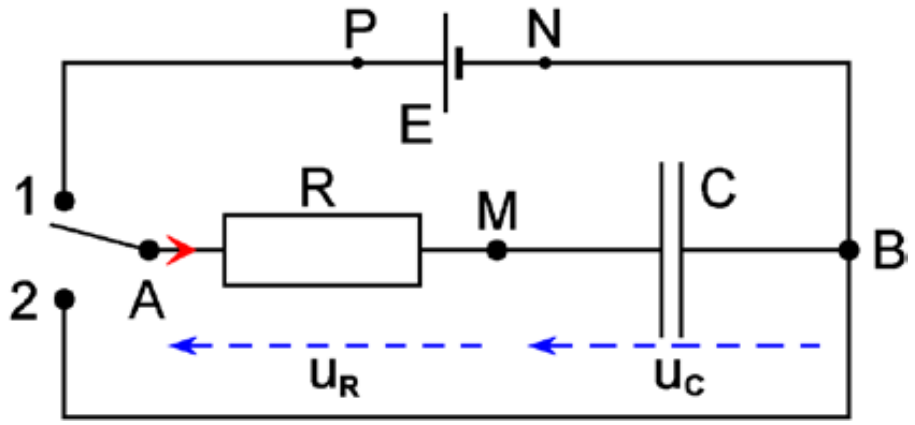
$$i_{AB} = \frac{dq_A}{dt} = C.\frac{d(V_A - V_B)}{dt} = C.\frac{du_{AB}}{dt}$$

On retiendra les relations suivantes :

$$u(t) = \frac{1}{C}.q(t) \quad \text{et} \quad i(t) = \frac{dq(t)}{dt} = C.\frac{du(t)}{dt}$$

. Le dipôle RC

Un dipôle RC est l'association en série d'un conducteur ohmique (conducteur ohmique) de résistance R et d'un condensateur de capacité C, comme dans le schéma ci-dessous :



1. Charge du condensateur

Le condensateur étant déchargé, on bascule, à l'instant t=0, l'interrupteur en position 1.

A chaque instant t > 0, on a :

$$\mathbf{V_A - V_B = u_R(t) + u_C(t) = R.i(t) + u_C(t) = E}$$

On a $i(t) = \frac{dq(t)}{dt}$ et $q(t) = C.u_C(t)$

donc $i(t) = C.\frac{du_C(t)}{dt}$ soit $R.C.\frac{du_C(t)}{dt} + u_C(t) = E$

D'où :

$$\frac{du_C(t)}{dt} + \frac{1}{R.C}.u_C(t) = \frac{E}{R.C}$$

On dit que $u_C(t)$ satisfait à une équation différentielle non homogène du premier ordre.

Avec $q(t) = C.u_C(t)$, on a :

$$\frac{dq(t)}{dt} + \frac{1}{R.C}.q(t) = \frac{E}{R}$$

A t = 0, le condensateur est déchargé et $q(0) = C.u_C(0) = 0$, on en déduit : $\left.\frac{du_C}{dt}\right)_{t=0} = \frac{E}{R.C}$

On a donc un point de la courbe représentative de $u_C(t)$: (0 ; 0) ainsi que la valeur de la pente de la tangente à cette courbe à l'origine des dates.

Au bout d'un temps assez long ($t = \infty$) on peut considérer que le condensateur est chargé et qu'il ne passe plus de charge dans le circuit : $\left.\frac{dq}{dt}\right)_{t=\infty} = C.\left.\frac{du_C}{dt}\right)_{t=\infty} = 0$ donc $\left.\frac{du_C}{dt}\right)_{t=\infty} = 0$

De l'équation différentielle, on tire : $u_C(\infty) = E$ tension aux bornes du générateur.

La courbe représentative de $u_C(t)$ tend vers la valeur E qui représente une asymptote avec une pente nulle. La courbe tend exponentiellement vers cette valeur.

Les solutions de l'équation différentielle sont de la forme :

$u_C(t) = A.e^{-m.t} + B$ où $m > 0$, m et B constantes d'intégration, A constante non définie.

Introduisons cette expression dans l'équation : $\frac{d(A.e^{-m.t}+B)}{dt} + \frac{1}{R.C}.(A.e^{-m.t} + B) = \frac{E}{R.C}$

D'où : $$-m.A.e^{-m.t} + \frac{A}{R.C}.e^{-m.t} + \frac{B}{R.C} = \frac{E}{R.C}$$

Cette équation doit être vérifiée à chaque instant, on en déduit : B = E

D'où : $$-m.A.e^{-m.t} + \frac{A}{R.C}.e^{-m.t} = 0 \quad \text{et} \quad m = \frac{1}{R.C}$$

La solution générale de l'équation différentielle s'écrit : $u_C(t) = A.e^{-\frac{t}{R.C}} + E$

A est une constante qui dépend des conditions initiales.

Ici, à t = 0, le condensateur est déchargé, donc : $u_C(0) = 0 = A.e^{-0} + E$ d'où $A = -E$

Compte tenu des conditions initiales imposées par l'expérience, la solution est :

$$u_C(t) = E.(1 - e^{-\frac{t}{R.C}})$$

2. Décharge du condensateur dans une résistance

Le condensateur étant chargé, on bascule, à l'instant t=0, l'interrupteur en position 2.

A chaque instant $t > 0$, on a : $V_A - V_B = u_R(t) + u_C(t) = R.i(t) + u_C(t) = 0$

On a $i(t) = \frac{dq(t)}{dt}$ et $q(t) = C.u_C(t)$ donc $i(t) = C.\frac{du_C(t)}{dt}$ soit $R.C.\frac{du_C(t)}{dt} + u_C(t) = 0$

D'où l'équation : $$\frac{du_C(t)}{dt} + \frac{1}{R.C}.u_C(t) = 0$$

On dit que $u_C(t)$ satisfait à une équation différentielle homogène du premier ordre.

Avec $q(t) = C.u_C(t)$, on a $$\frac{dq(t)}{dt} + \frac{1}{R.C}.q(t) = 0$$

A t = 0, le condensateur est chargé $q(0) = C.u_C(0) = C.E$, on en déduit : $\left.\frac{du_C}{dt}\right)_{t=0} = -\frac{E}{R.C}$

On a donc un point de la courbe représentative de $u_C(t)$: (0 ; E) ainsi que la valeur de la pente de la tangente à cette courbe à l'origine des dates.

Au bout d'un temps assez long ($t = \infty$) on peut considérer que le condensateur est déchargé et qu'il ne passe plus de charge dans le circuit : $\left.\frac{dq}{dt}\right)_{t=\infty} = C.\left.\frac{du_C}{dt}\right)_{t=\infty} = 0$ donc

$\left.\frac{du_C}{dt}\right)_{t=\infty} = 0$ et $u_C(\infty) = 0$.

La courbe représentative de $u_C(t)$ tend vers 0 qui représente une asymptote avec une pente nulle. La courbe tend exponentiellement vers 0.

Les solutions de l'équation différentielle sont de la forme :

$u_C(t) = A.e^{-m.t} + B$ où $m > 0$, m et B constantes d'intégration, A constante non définie.

Introduisons cette expression dans l'équation : $\frac{d(A.e^{-m.t}+B)}{dt} + \frac{1}{R.C}.(A.e^{-m.t} + B) = 0$

D'où : $$-m.A.e^{-m.t} + \frac{A}{R.C}.e^{-m.t} + \frac{B}{R.C} = 0$$

Cette équation doit être vérifiée à chaque instant, on en déduit : B = 0

D'où : $$-m.A.e^{-m.t} + \frac{A}{R.C}.e^{-m.t} = 0 \text{ et } m = \frac{1}{R.C}$$

La solution générale de l'équation différentielle s'écrit : $u_C(t) = A.e^{-\frac{t}{R.C}}$

Là encore A est une constante qui dépend des conditions initiales.

Ici, à t = 0, le condensateur est chargé, donc : $u_C(0) = E = A.e^{-0}$ d'où $A = E$

Compte tenu des conditions initiales imposées par l'expérience, la solution est :

$$u_C(t) = E.e^{-\frac{t}{R.C}}$$

En appliquant les relations : $q(t) = C.u_C(t)$ et $i(t) = C.\frac{du_C(t)}{dt}$

On a : $q(t) = C.E.\ e^{-\frac{t}{R.C}}$ et $i(t) = -\frac{E}{R}.e^{-\frac{t}{R.C}}$

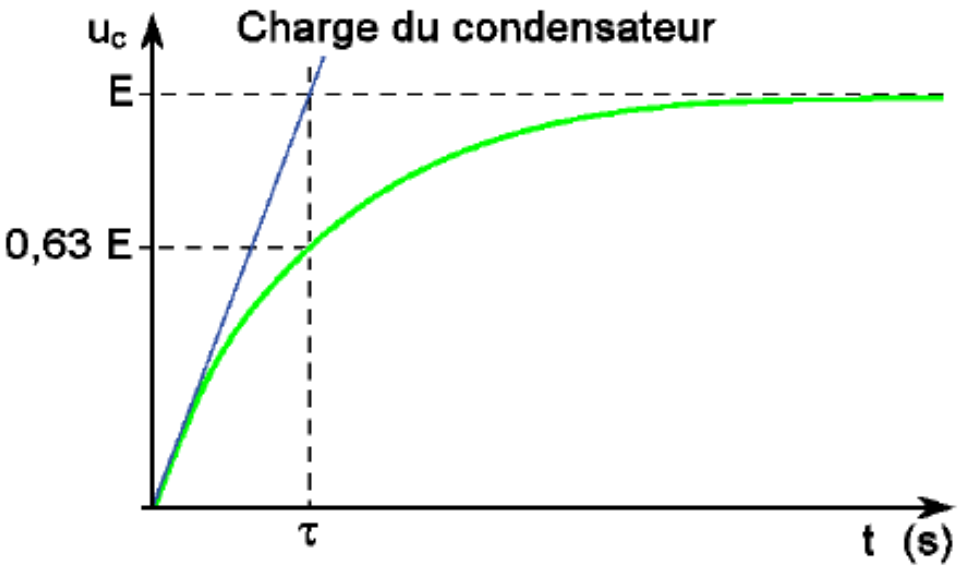
Au bout d'un temps long : $q(\infty) = 0$ le condensateur est déchargé
Et : $i(\infty) = 0$ il n'y a plus de courant

3. Constante de temps du dipôle RC

Le produit R.C est homogène à un temps et est appelé constante de temps τ du dipôle RC.

. Lors de la charge :

$$u_C(t) = E.(1 - e^{-\frac{t}{R.C}})$$



La tangente à la courbe à l'origine coupe l'asymptote y = E au point d'abscisse t = τ.
En effet, la tangente à la courbe représentative de $u_C(t)$, à l'origine des dates, a pour équation :

$$y = \left.\frac{d[u_C(t)]}{dt}\right)_{t=0}.t = -E.(-\frac{1}{R.C}).t = \frac{E}{R.C}.t$$

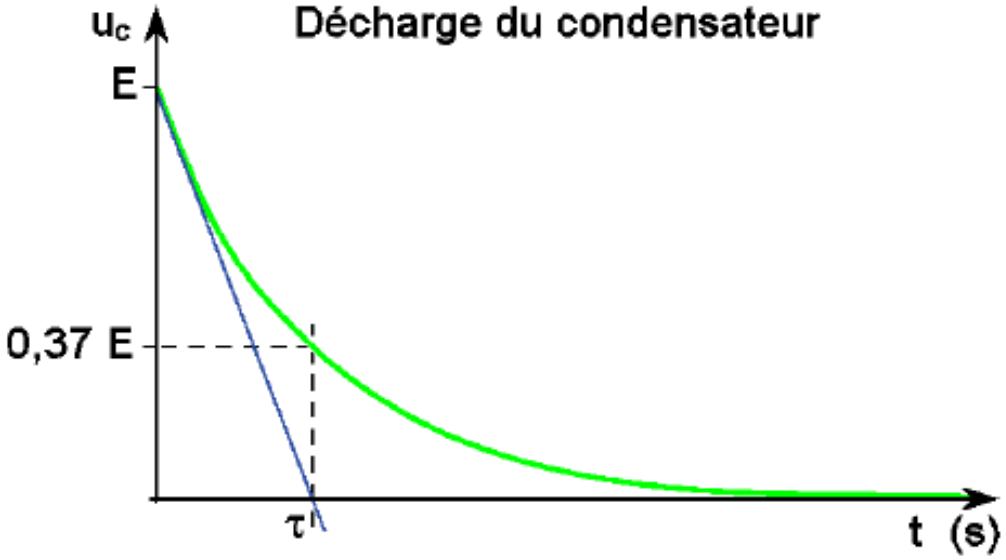
Elle coupe l'asymptote y = E en un point d'abscisse t : E/(R.C).t = E soit $t = R.C = \tau$

Et : $u_C(\tau) = E.(1 - e^{-1}) \approx 0,63.E.$

Par lecture graphique de l'abscisse du point de la courbe dont l'ordonnée est égale à 0,63.E, on obtient la valeur de τ.

. **Lors de la décharge :**

$$u_C(t) = E.e^{-\frac{t}{R.C}}$$



La tangente à la courbe, à t=0s, coupe l'asymptote y=0 au point d'abscisse t = τ.

En effet, la tangente à la courbe représentative de $u_c(t)$, à t = 0s, a pour équation :

$$y = E + \left.\frac{d[u_C(t)]}{dt}\right)_{t=0}.t = E - \frac{E}{R.C}.t$$

Si t = τ = R.C , on a alors : y = 0

On a également :

$$u_C(\tau) = E.e^{-1} \approx 0,37.E$$

Par lecture graphique de l'abscisse du point de la courbe dont l'ordonnée est égale à 0,37.E, on obtient la valeur de τ.

. Firmata Express

Pour contrôler l'Arduino via le protocole de communication Firmata Express, le programme Python utilise la bibliothèque **"pymata-express"** (le sketch **"Firmata Express"** doit être téléversé dans la mémoire de l'Arduino au préalable).

Toutes les fonctions utiles à l'utilisation de **"Pymata-express"** (fonction de déclaration des entrées et sorties, de lectures, d'écritures…) ont été regroupées dans un fichier Python nommé **"PymataExpressDef.Py"** que l'on peut importer dans tous les programmes, à condition que le fichier des fonctions soit dans le même dossier que le fichier du programme, avec l'instruction :

from PymataExpressDef import *

```
import asyncio


########## DEFINITION D'UNE BOUCLE ASYNCIO ##########

loop = asyncio.get_event_loop()


########## GESTION DES ENTREES NUMERIQUES ##########

# DECLARATION D'UNE BROCHE EN ENTREE NUMERIQUE

def Set_DigitalInput_Pin(board, pin):
    loop.run_until_complete(board.set_pin_mode_digital_input(pin))

# LECTURE DE L'ETAT LOGIQUE D'UNE BROCHE DECLAREE EN ENTREE NUMERIQUE

def Digital_Read(board, pin):
    value = loop.run_until_complete(board.digital_read(pin))
    return value[0]


########## GESTION DES SORTIES NUMERIQUES ##########

# DECLARATION D'UNE BROCHE EN SORTIE NUMERIQUE

def Set_DigitalOutput_Pin(board, pin):
    loop.run_until_complete(board.set_pin_mode_digital_output(pin))

# MODIFICATION DE L'ETAT LOGIQUE D'UNE SORTIE NUMERIQUE

def Digital_Write(board, pin, val):
    loop.run_until_complete(board.digital_write(pin, val))


########## GESTION DES ENTREES ANALOGIQUES ##########

# DECLARATION D'UNE BROCHE EN ENTREE ANALOGIQUE

def Set_AnalogInput_Pin(board,pin):
    loop.run_until_complete(board.set_pin_mode_analog_input(pin))

# LECTURE DE LA VALEUR DE L'ENTREE ANALOGIQUE

def Analog_Read(board, pin):
    value = loop.run_until_complete(board.analog_read(pin))
    return value[0]
```

```
########## GESTION DES SORTIES ANALOGIQUES ##########

# DECLARATION D'UNE BROCHE EN SORTIE ANALOGIQUE

def Set_AnalogOutput_Pin(board,pin):
    loop.run_until_complete(board.set_pin_mode_pwm(pin))

# MODIFICATION DE LA VALEUR D'UNE SORTIE ANALOGIQUE

def Analog_Write(board, pin, val):
    loop.run_until_complete(board.analog_write(pin, val))


########## GESTION DU SON ##########

# DECLARATION D'UNE BROCHE EN MODE TONE

def Set_Tone_Pin(board,pin):
    loop.run_until_complete(board.set_pin_mode_tone(pin))

# EMISSION SONORE

def Tone(board,pin,freq,duration):
    if duration>0:
        loop.run_until_complete(board.play_tone(pin, freq, duration))
    else:
        loop.run_until_complete(board.play_tone_continuously(pin, freq))

# ARRET DE L'EMISSION SONORE

def No_Tone(board,pin):
    loop.run_until_complete(board.play_tone_off(pin))


########## DECONNEXION DE L'ARDUINO ##########

def Arduino_Exit(board):
    loop.run_until_complete(board.shutdown())
```

. Connexion à l'Arduino avec le protocole de communication "Firmata Express"

Dans les programmes de contrôle de l'Arduino par le protocole de communication "Firmata Express", le module "**ConnectToArduino.py**", contenant les fonctions de connexion avec l'Arduino, sera importé.

La connexion se déroulera en 2 étapes :

- Appel de la fonction de sélection du port COM du module "**ConnectToArduino.py**":

  PortComArduino = SelectPortCOM()

  Le nombre de port COM disponible est alors déterminé :

  PortsCOM = list(serial.tools.list_ports.comports()

  –> si nombre de port COM = 0 : message d'erreur,

  –> si nombre de port COM = 1 : sélection de ce port COM pour la connexion,

  –> si nombre de port COM > 1 : L'utilisateur doit sélectionner le bon port COM.

```
def SelectPortCOM():

    Nport=[]
    PortsCOM = list(serial.tools.list_ports.comports())
    for port_numero, description, address in PortsCOM:
        Nport.append(port_numero)


    if len(PortsCOM)== 0:
        print("Aucune carte Arduino n'a été détectée!")
        saisie = ""
        while saisie != "q":
            saisie = str(input("Entrez 'q' pour quitter: "))
        sys.exit()

    elif len(PortsCOM)== 1:
        PortComArduino = Nport[0]

    else:
        print("Liste des ports COM disponibles:\n")
        for i in range(len(PortsCOM)):
            print(i+1, ": ", PortsCOM[i])
        ChoixPort=False
        while ChoixPort==False:
            Choix = input("\nVeuillez indiquer le numéro du port de la carte Arduino:")
            try:
                Choix = int(Choix)
                assert Choix >= 1 and Choix <= len(PortsCOM)
                ChoixPort = True
            except AssertionError:
                print("Le numéro indiqué n'est pas entre 1 et", len(PortsCOM) ,"!")
                ChoixPort = False
            except:
                print("Vous n'avez pas saisi un numéro entre 1 et", len(PortsCOM) ,"!")
        PortComArduino = Nport[Choix-1]

    return PortComArduino
```

- Tentative d'ouverture du port COM sélectionné (**PortComArduino**) et de connexion à l'Arduino via le protocole **"Firmata Express"** avec la fonction **"OpenPortCom"** du module **"ConnectToArduino.py"** :

board = OpenPortCom(PortComArduino)

```
def OpenPortCom(PortCom):

    try:
        board = PymataExpress(com_port = PortCom)

    except:
        AffichMessageErreur(PortCom)

    else:
        return board


def AffichMessageErreur(PortCom):

    print("Un problème s'est produit à l'ouverture du port.\n"
        "Vérifiez que le port utilisé par la carte Arduino est bien "+ PortCom + ",\n" +
        "et que le protocole de communication FIRMATA EXPRESS a bien été téléversé.\n")
    saisie = ""
    while saisie != "q":
        saisie = str(input("Entrez 'q' pour quitter: "))
    sys.exit()
```

- Activité 1 : Etude de la charge d'un condensateur d'un dipôle RC

. Objectif

L'objectif de l'activité est de suivre l'évolution temporelle de la tension aux bornes du condensateur lors de sa charge afin de vérifier la relation :

$$u_C(t) = E.(1 - e^{-\frac{t}{R.C}})$$

. Descriptif de l'activité

Après avoir déchargé le condensateur, la mesure de la tension aux bornes du condensateur Uc, lors de la charge, à l'aide de l'entrée analogique A0 est lancée, à t = 0 s, par un appui sur le bouton poussoir.

La valeur de la tension en V est affichée dans la console Python ou le moniteur série toutes les 100 ms.

Les mesures sont arrêtées en appuyant sur le bouton poussoir. Le condensateur est alors déchargé afin de pouvoir effectuer de nouvelles mesures en appuyant de nouveau sur le bouton poussoir.

Il est donc possible d'acquérir des couples de données (t, Uc) afin de vérifier la relation Uc = f(t) théorique.

. Le programme

Voici le code de l'activité en Python et en langage Arduino :

. Programme en Python ("Projet7\Activity1\PY\Activity1.py")

```
# Importations des librairies et définition de fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time

def decharge(board, pinalim, pinU):
    Digital_Write(board, pinalim, 0);
    print("Décharge du condensateur.")
    while Analog_Read(board, pinU) > 0:
        time.sleep(0.2)
    print("Condensateur déchargé.")


# Déclaration des constantes et variables

PinUC = 0
PinButton = 12
PinAlimC = 2

ValPinUC = 0
UC = 0.0
t0 = 0
dt = 0

ValButton = 0
OldValButton = 0
State = 0
OldState = 0


# Connexion à l'Arduino

print("\nConnexion à l'Arduino en cours...")

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)

Set_AnalogInput_Pin(board, PinUC)
Set_DigitalInput_Pin(board, PinButton)
Set_DigitalOutput_Pin(board, PinAlimC)

print("Connexion à l'Arduino établie - Appuyez sur Ctrl-C pour quitter.\n")

decharge(board, PinAlimC, PinUC)

print("\nAppuyez sur le bouton poussoir pour commencer les mesures.\n")


# Boucle principale du programme

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        time.sleep(0.01)
        if ValButton == 1 and OldValButton == 0:
            State=1-State

        OldValButton = ValButton;

        if State==1:
            if OldState == 0:
               print("\nCharge du condensateur en cours.\n")
               print("Temps (S); Uc (V):")
               t0 = time.time()
               Digital_Write(board, PinAlimC, 1)
               OldState=1
```

```
            ValPinUC = Analog_Read(board, PinUC)
            UC = (ValPinUC/1023)*5.0
            dt = time.time()-t0
            print(round(dt,2),";",round(UC,2))

            time.sleep(0.1)

        else:
            if OldState == 1:
                print("\nFin des mesures.\n")
                decharge(board, PinAlimC, PinUC)
                print("\nAppuyez sur le bouton poussoir pour commencer les mesures.\n")
                OldState = 0

    except KeyboardInterrupt:
        print("\nFin du programme.\n")
        decharge(board, PinAlimC, PinUC)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **"ConnectToArduino.py",** contenant les fonctions de connexion à l'Arduino via le protocole **"Firmata Express"**,

. Le module **"PymataExpressDef.Py"** regroupant toutes les fonctions utiles à l'utilisation de **"Pymata-express"** (fonction de déclaration des entrées et sorties, de lectures, d'écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause,

. La fonction **"décharge"** pour décharger le condensateur avant l'étude de la charge.

- Déclaration des constantes et variables :

. **PinUc = 0** (cst correspondant au n° de la broche A0 sur laquelle le condensateur est connecté)

. **PinAlimC = 2** (cst correspondant au n° de la broche alimentant le dipôle RC)

. **ValPinUC = 0** (variable pour stocker la valeur de la broche du condensateur)

. **Uc = 0.0** (variable pour stocker le résultat du calcul de la tension aux bornes du condensateur)

. **t0 = 0** (variable pour stocker le temps de début de charge du condensateur)

. **dt = 0** (variable correspondant à la différence de temps en s entre les mesures de tension du condensateur et le temps de début de charge)

. **PinButton = 12** (cst correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

. **ValButton = 0** (variable pour stocker la valeur de l'état logique de la broche du bouton poussoir)

. **OldValButton = 0** (variable pour stocker la valeur précédente de l'état logique de la broche du bouton poussoir)

. **State=0** (variable correspondant à l'action à effectuer)

. **OldState = 0** (variable pour stocker la valeur précédente de la variable **State**)

. **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d'ouverture du port COM sélectionné et connexion à l'Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l'Arduino est réussie :

- Déclaration de la broche du condensateur en entrée analogique :

Set_AnalogInput_Pin(board, PinUC)

- Déclaration de la broche du bouton poussoir en entrée numérique :

Set_DigitalInput_Pin(board, PinButton)

- Déclaration de la broche alimentant le dipôle RC en sortie numérique :

Set_DigitalOutput_Pin(board, PinAlimC)

- Décharge du condensateur :

--> Appel de la fonction **"decharge"** : **decharge(board, PinAlimC, PinUC)**

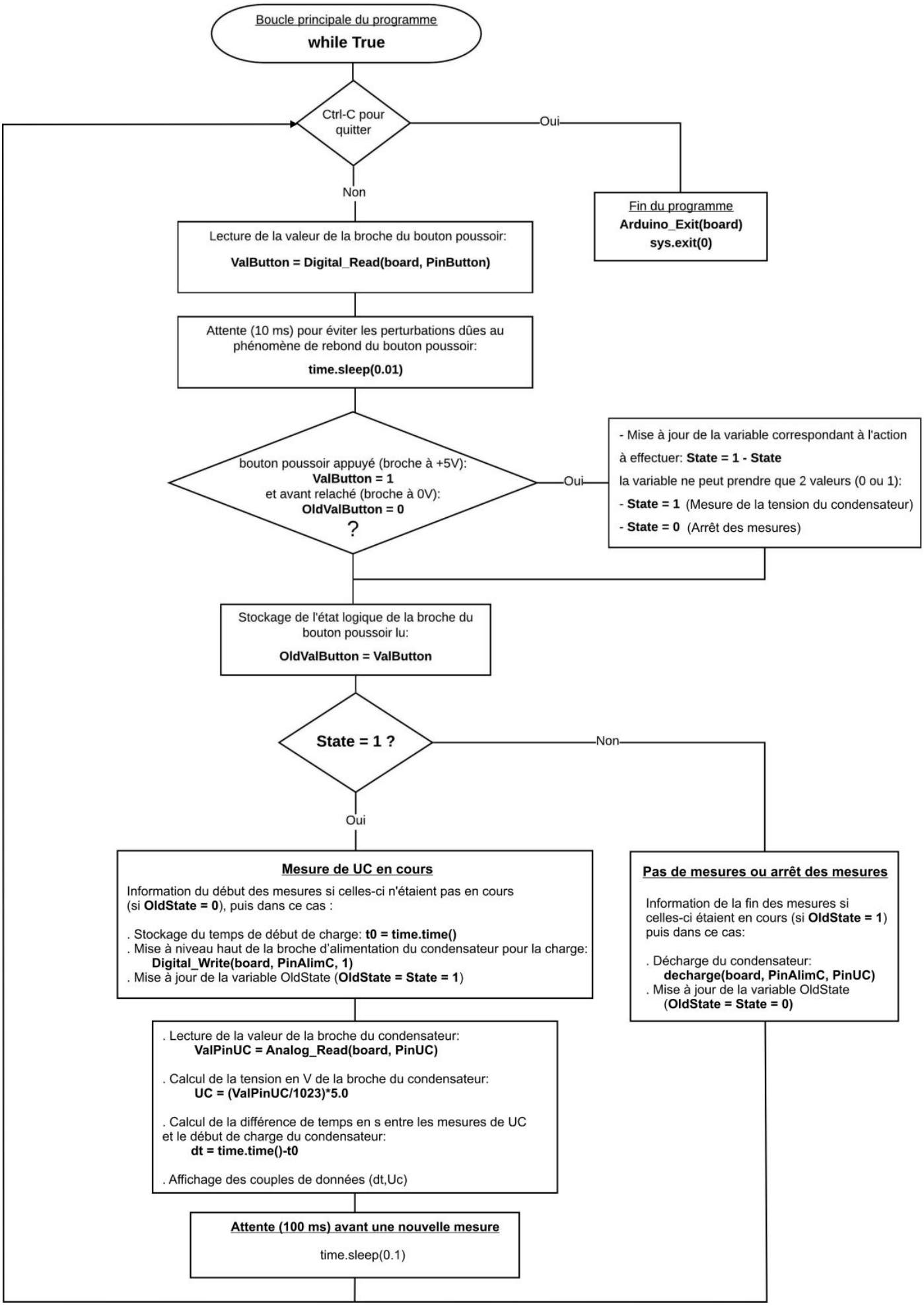
. Mise à niveau bas de la broche d'alimentation du dipôle RC:

Digital_Write(board, pinalim, 0)

. Attente de la décharge totale du condensateur:

```
while Analog_Read(board, pinU) > 0:
    time.sleep(0.2)
```

- Boucle principale du programme (boucle "while True") :


Boucle principale du programme
while True
Ctrl-C pour quitter
Oui
Non
Fin du programme
Arduino_Exit(board)
sys.exit(0)
Lecture de la valeur de la broche du bouton poussoir:
ValButton = Digital_Read(board, PinButton)
Attente (10 ms) pour éviter les perturbations dûes au phénomène de rebond du bouton poussoir:
time.sleep(0.01)
bouton poussoir appuyé (broche à +5V):
ValButton = 1
et avant relaché (broche à 0V):
OldValButton = 0
?
Oui
- Mise à jour de la variable correspondant à l'action à effectuer: State = 1 - State
la variable ne peut prendre que 2 valeurs (0 ou 1):
- State = 1 (Mesure de la tension du condensateur)
- State = 0 (Arrêt des mesures)
Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Non
Oui
Mesure de UC en cours
Information du début des mesures si celles-ci n'étaient pas en cours (si OldState = 0), puis dans ce cas :
. Stockage du temps de début de charge: t0 = time.time()
. Mise à niveau haut de la broche d'alimentation du condensateur pour la charge:
Digital_Write(board, PinAlimC, 1)
. Mise à jour de la variable OldState (OldState = State = 1)
Pas de mesures ou arrêt des mesures
Information de la fin des mesures si celles-ci étaient en cours (si OldState = 1) puis dans ce cas:
. Décharge du condensateur:
decharge(board, PinAlimC, PinUC)
. Mise à jour de la variable OldState
(OldState = State = 0)
. Lecture de la valeur de la broche du condensateur:
ValPinUC = Analog_Read(board, PinUC)
. Calcul de la tension en V de la broche du condensateur:
UC = (ValPinUC/1023)*5.0
. Calcul de la différence de temps en s entre les mesures de UC et le début de charge du condensateur:
dt = time.time()-t0
. Affichage des couples de données (dt,Uc)
Attente (100 ms) avant une nouvelle mesure
time.sleep(0.1)

Résultats dans la fenêtre Python Shell

```
Connexion à l'Arduino établie - Appuyez sur Ctrl-C pour quitter.

Décharge du condensateur.
Condensateur déchargé.

Appuyez sur le bouton poussoir pour commencer les mesures.


Charge du condensateur en cours.

Temps (S); Uc (V):
0.0 ; 0.0
0.14 ; 0.57
0.28 ; 1.18
0.42 ; 1.64
0.56 ; 2.04
0.71 ; 2.44
0.84 ; 2.74
0.99 ; 3.04
1.13 ; 3.27
1.29 ; 3.52
1.45 ; 3.72
1.58 ; 3.87
1.72 ; 4.0
1.86 ; 4.13
2.0 ; 4.23

Fin des mesures.

Décharge du condensateur.
Condensateur déchargé.

Appuyez sur le bouton poussoir pour commencer les mesures.
```

. Programme en langage Arduino (”Projet7\Activity1\INO\Activity1.ino”)

Activity1

```
// Déclaration des constantes et variables

const int PinUC = 0;
const int PinButton = 12;
const int PinAlimC = 2;

int ValPinUC = 0;
float UC = 0.0;
unsigned long t0;
float dt;

int ValButton = 0;
int OldValButton = 0;
int State = 0;
int OldState = 0;


// Déclaration de fonctions

void decharge(){
  digitalWrite(PinAlimC, LOW);
  Serial.println("Decharge du condensateur");
  while (analogRead(PinUC) > 0) {
    delay(200);
    }
  Serial.println("Condensateur decharge");
}


// Initialisation des entrées et sorties

void setup() {
  Serial.begin(9600);
  pinMode(PinButton, INPUT);
  pinMode(PinAlimC, OUTPUT);
  decharge();
  Serial.println("Appuyez sur le bouton poussoir pour commencer les mesures.");
}


// Fonction principale en boucle

void loop() {
  ValButton = digitalRead(PinButton);
delay(10);

if ((ValButton == HIGH)&&(OldValButton == LOW))
  {
  State=1-State;
  }
OldValButton = ValButton;
```

```
if (State==1)
  {
  if (OldState == 0)
  {
   Serial.println("Charge du condensateur en cours.");
   Serial.println("");
   Serial.println ("Temps (S);Uc (V):");
   t0 = micros();
   digitalWrite(PinAlimC, 1);
   OldState=1;
  }
  ValPinUC = analogRead(PinUC);
  UC = (ValPinUC/1023.0)*5.0;
  dt =  (micros() - t0)* 1e-6;
  Serial.print(dt,2);
  Serial.print(";");
  Serial.println(UC,2);

  delay(100);
  }
  else
  {
    if (OldState == 1){
    Serial.println("Fin des mesures.");
    decharge();
    OldState = 0;}
  }
}
```

Déroulement du programme :

- Déclaration des constantes et variables :

. **const int PinUc = 0** (broche du condensateur : A0)
. const int PinButton = 12 (broche du bouton poussoir)
. **const int PinAlimC = 2** (broche d'alimentation du dipôle RC)

. **int ValPinUc = 0** (variable nombre entier valeur broche du condensateur)
. **float Uc = 0.0** (variable nombre décimal calcul tension Uc)
. unsigned long t0 (variable nombre entier long temps début charge)
. float dt (variable nombre décimal différence de temps entre les mesures de UC et le temps de début de charge)

. int ValButton = 0 (variable nombre entier valeur broche bouton poussoir)
. int OldValButton = 0 (variable nombre entier ancienne valeur broche bouton poussoir)
. int State = 0 (variable nombre entier pour action à effectuer)
. int OldState = 0 (variable nombre entier pour action effectuée précédemment)

- Déclaration de fonctions :

→ Fonction permettant de décharger le condensateur :

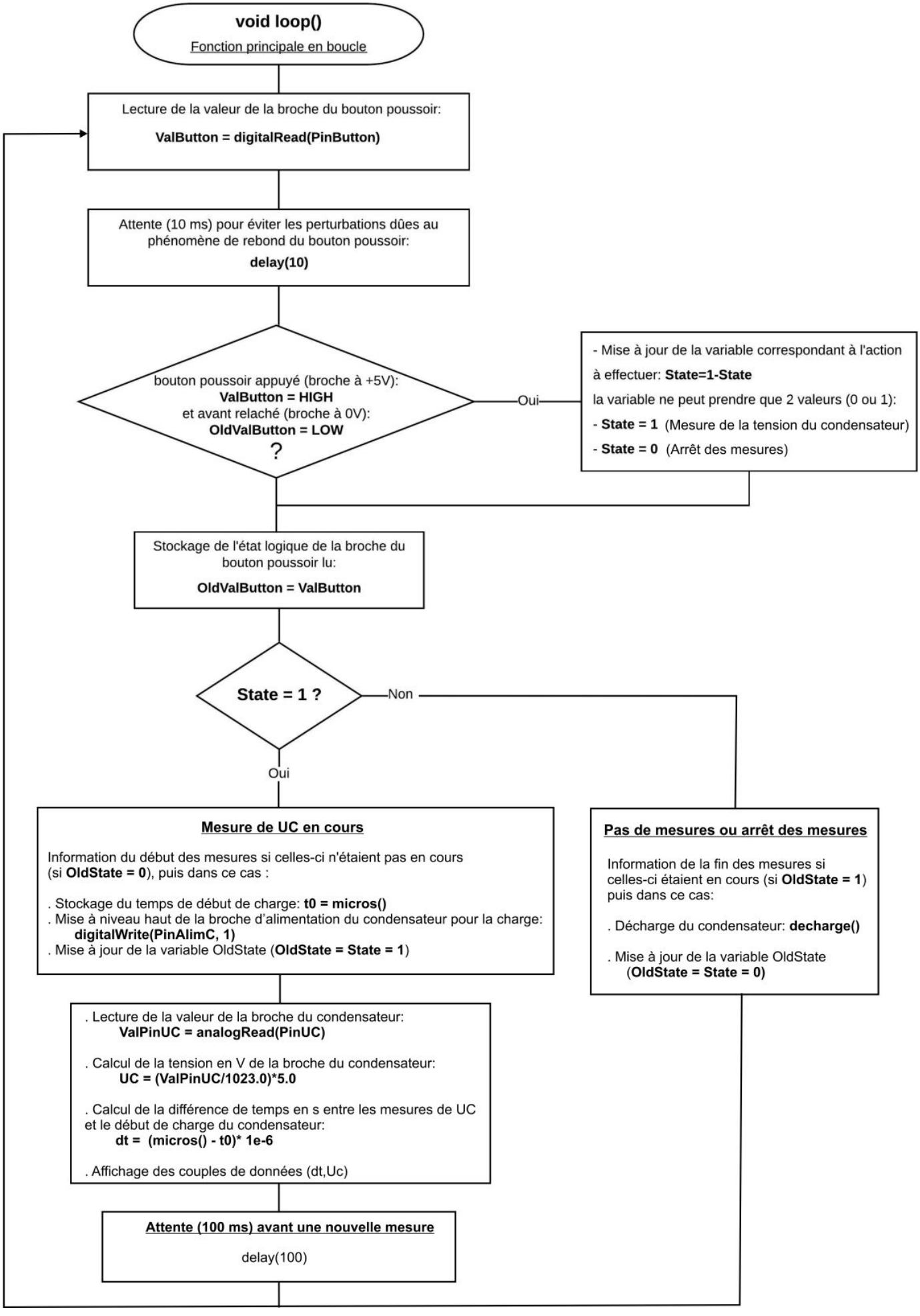
- Mise à niveau bas de la broche d'alimentation du dipôle RC :
  digitalWrite(PinAlimC, LOW)

- Attente de la fin de la décharge du condensateur :
  while (analogRead(PinUC) > 0) ;
  delay(200) ;

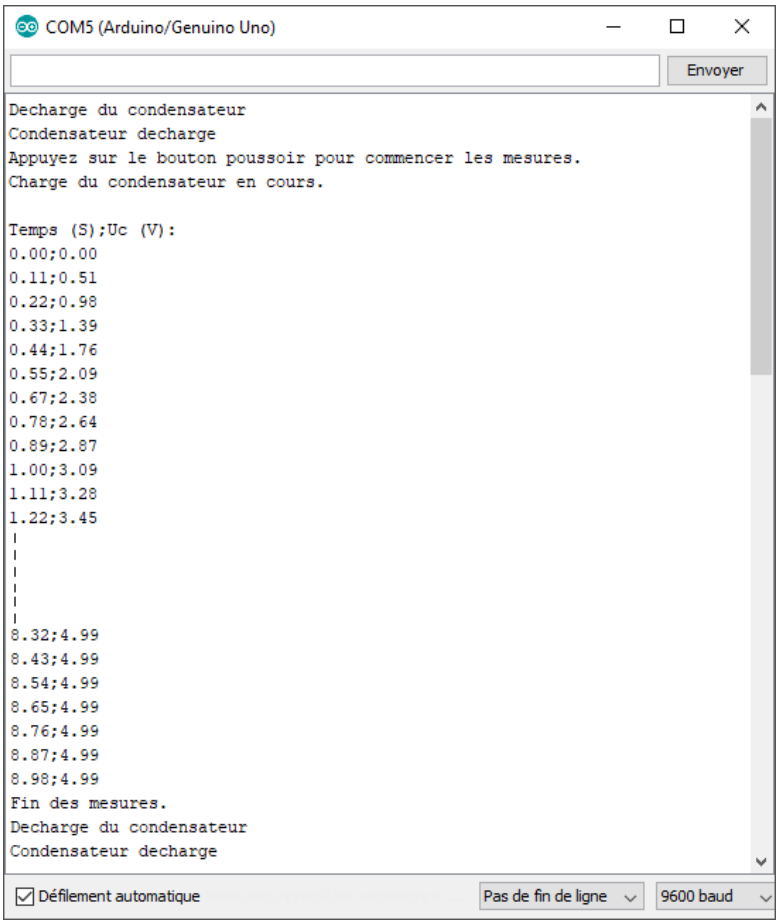
- Initialisation des entrées et sorties :

. Initialisation de la liaison série à un débit de 9600 bauds,
. Initialisation de la broche du bouton poussoir en entrée numérique,
. Initialisation de la broche d'alimentation du dipôle RC en sortie numérique,
. Décharge du condensateur.

- Fonction principale en boucle :

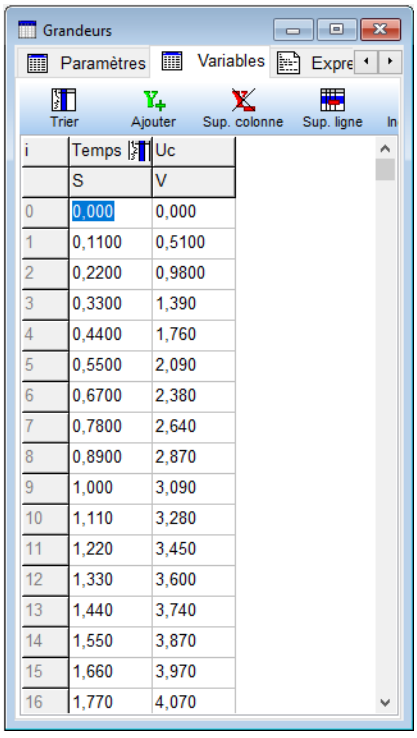

void loop()
Fonction principale en boucle
Lecture de la valeur de la broche du bouton poussoir:
ValButton = digitalRead(PinButton)
Attente (10 ms) pour éviter les perturbations dûes au phénomène de rebond du bouton poussoir:
delay(10)
bouton poussoir appuyé (broche à +5V):
ValButton = HIGH
et avant relaché (broche à 0V):
OldValButton = LOW
?
Oui
- Mise à jour de la variable correspondant à l'action à effectuer: State=1-State
la variable ne peut prendre que 2 valeurs (0 ou 1):
- State = 1 (Mesure de la tension du condensateur)
- State = 0 (Arrêt des mesures)
Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Non
Oui
Mesure de UC en cours
Information du début des mesures si celles-ci n'étaient pas en cours (si OldState = 0), puis dans ce cas :
. Stockage du temps de début de charge: t0 = micros()
. Mise à niveau haut de la broche d'alimentation du condensateur pour la charge:
digitalWrite(PinAlimC, 1)
. Mise à jour de la variable OldState (OldState = State = 1)
Pas de mesures ou arrêt des mesures
Information de la fin des mesures si celles-ci étaient en cours (si OldState = 1) puis dans ce cas:
. Décharge du condensateur: decharge()
. Mise à jour de la variable OldState
(OldState = State = 0)
. Lecture de la valeur de la broche du condensateur:
ValPinUC = analogRead(PinUC)
. Calcul de la tension en V de la broche du condensateur:
UC = (ValPinUC/1023.0)*5.0
. Calcul de la différence de temps en s entre les mesures de UC et le début de charge du condensateur:
dt = (micros() - t0)* 1e-6
. Affichage des couples de données (dt,Uc)
Attente (100 ms) avant une nouvelle mesure
delay(100)

Résultats dans le moniteur série :



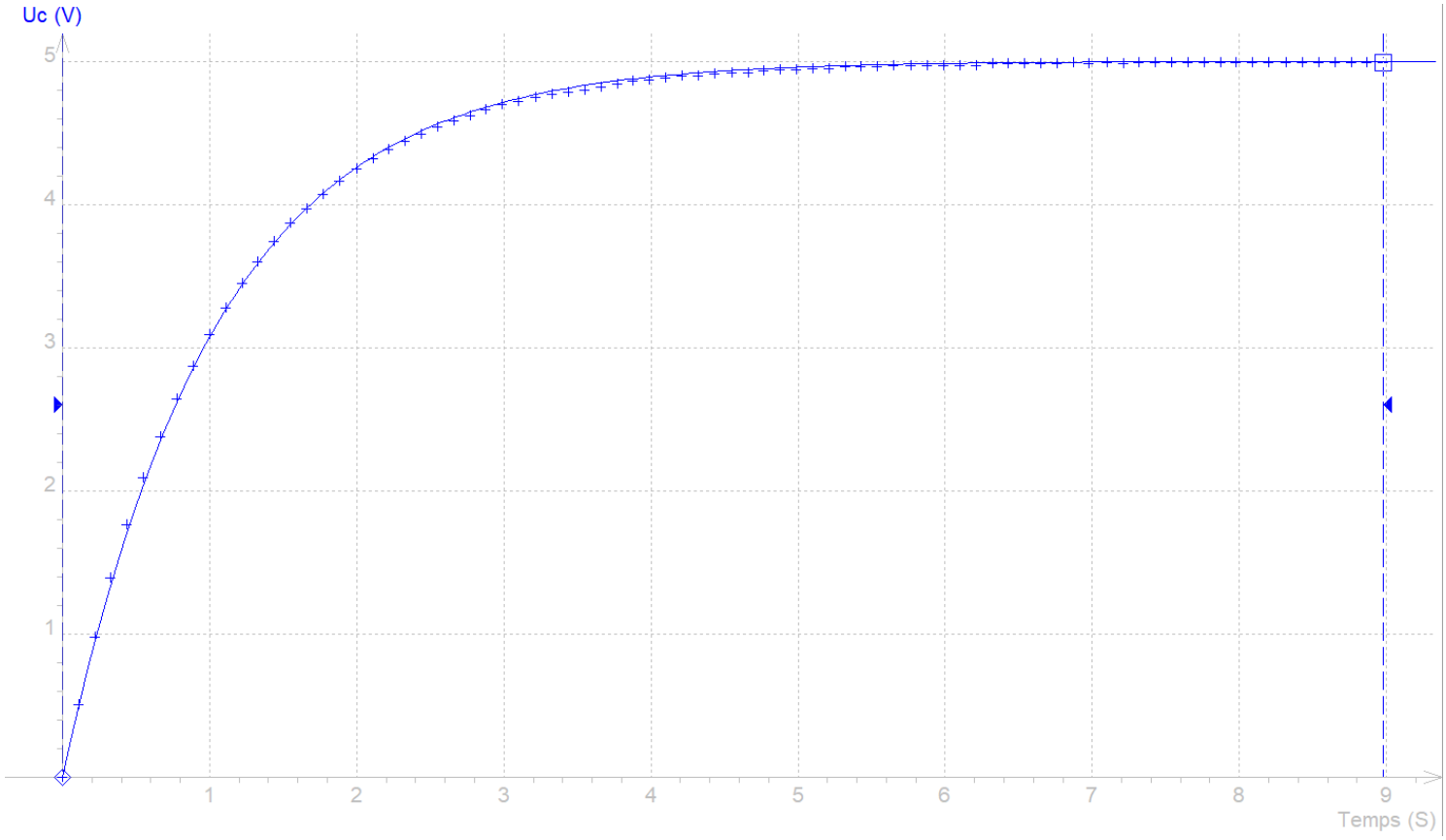
. Exploitation des mesures :

Pour exploiter les mesures, il suffit de sélectionner et de copier toutes les données à partir de la fenêtre Python Shell ou du moniteur série et d'ouvrir un nouveau fichier dans Regressi à partir du "Presse-papier".



i	Temps	Uc
	S	V
0	0,000	0,000
1	0,1100	0,5100
2	0,2200	0,9800
3	0,3300	1,390
4	0,4400	1,760
5	0,5500	2,090
6	0,6700	2,380
7	0,7800	2,640
8	0,8900	2,870
9	1,000	3,090
10	1,110	3,280
11	1,220	3,450
12	1,330	3,600
13	1,440	3,740
14	1,550	3,870
15	1,660	3,970
16	1,770	4,070

On peut alors tracer le graphe représentant la tension aux bornes du condensateur en fonction du temps :



Et effectuer une modélisation suivant :

$$u_C(t) = E.(1 - e^{-\frac{t}{R.C}})$$

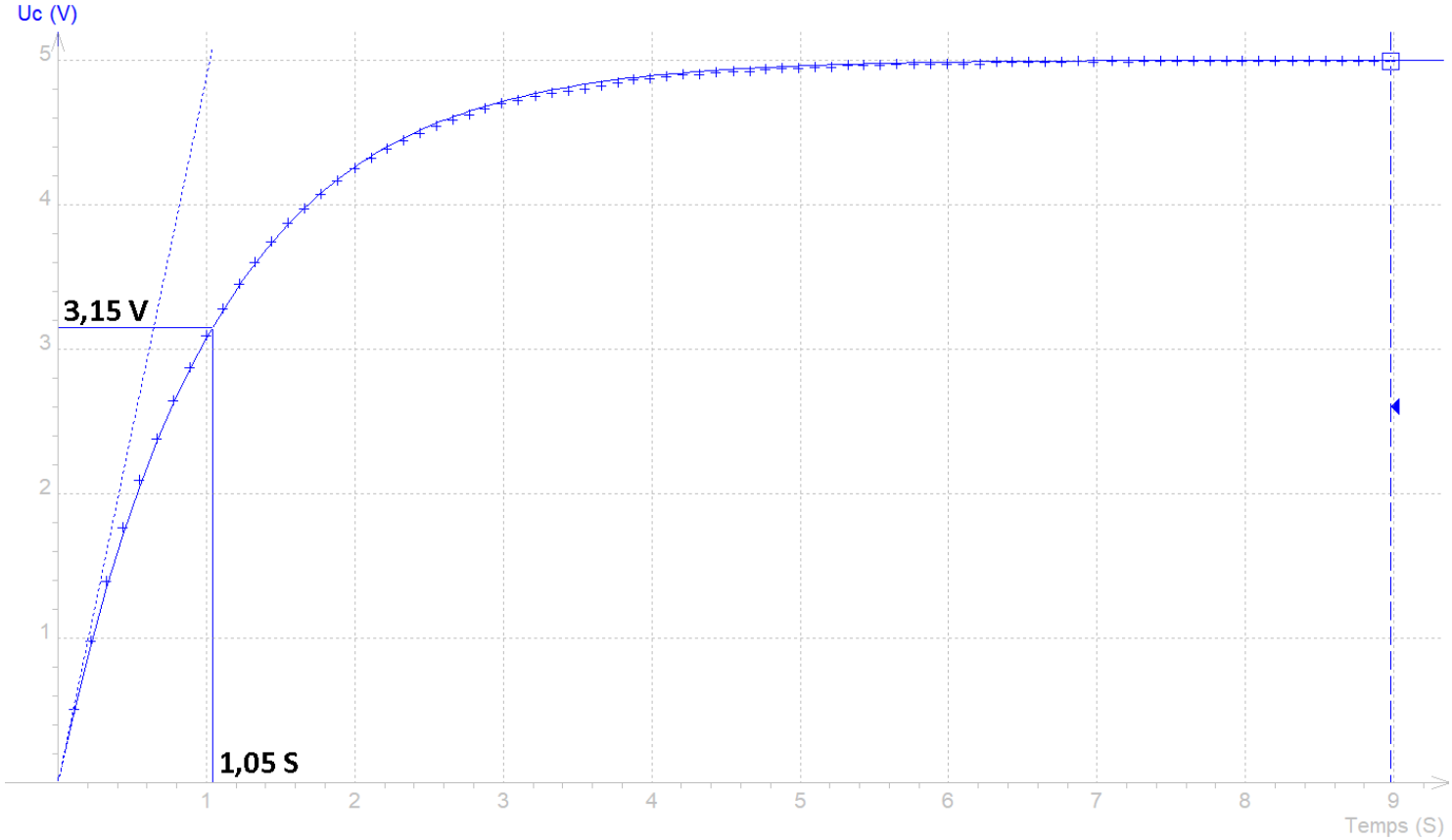
Expression du modèle	Résultats de la modélisation
Uc(Temps)=5*(1-exp(-Temps/tau))	Ecart expérience-modèle 0,38 % sur Uc(Temps) Ecart quad. Uc=17,13 mV tau=1,048 ±0,005

La valeur théorique de τ est :

$$\tau = RC = 10.10^3 \times 100.\ 10^{-6} = 1\ S$$

Par la modélisation, la détermination de τ donne une valeur très proche de la valeur théorique.

On peut également déterminer τ à l'aide de la tangente à l'origine ou par la mesure du temps pour lequel le condensateur est chargé à 63 % (0,63 x 5 = 3,15 V):



- Activité 2 : Etude de la décharge d'un condensateur d'un dipôle RC

. Objectif

L'objectif de l'activité est de suivre l'évolution temporelle de la tension aux bornes du condensateur lors de sa décharge, avec le même circuit que l'activité précédente, afin de vérifier la relation :

$$u_C(t) = E.e^{-\frac{t}{R.C}}$$

. Descriptif de l'activité

Après avoir chargé le condensateur, la mesure de la tension aux bornes du condensateur U_c, lors de la décharge, à l'aide de l'entrée analogique A0 est lancée, à t=0s, par un appui sur le bouton poussoir.

La valeur de la tension en V est affichée dans la fenêtre Python Shell ou le moniteur série toutes les 100 ms.

Les mesures sont arrêtées en appuyant sur le bouton poussoir. Le condensateur est alors chargé afin de pouvoir effectuer de nouvelles mesures en appuyant de nouveau sur le bouton poussoir.

Il est donc possible d'acquérir des couples de données (t, U_c) afin de vérifier la relation U_c=f(t) théorique.

. Le programme

Voici le code de l'activité en Python et en langage Arduino :

. Programme en Python ("Projet7\Activity2\PY\Activity2.py")

```
# Importations des librairies et définition de fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time

def charge(board, pinalim, pinU):
    Digital_Write(board, pinalim, 1);
    print("Charge du condensateur.")
    while Analog_Read(board, pinU) < 1023:
        time.sleep(0.2)

    print("Condensateur chargé.")

def decharge(board, pinalim, pinU):
    Digital_Write(board, pinalim, 0);
    print("Décharge du condensateur.")
    while Analog_Read(board, pinU) > 0:
        time.sleep(0.2)
    print("Condensateur déchargé.")


# Déclaration des constantes et variables

PinUC = 0
PinButton = 12
PinAlimC = 2

ValPinUC = 0
UC = 0.0
t0 = 0
dt = 0

ValButton = 0
OldValButton = 0
State = 0
OldState = 0


# Connexion à l'Arduino

print("\nConnexion à l'Arduino en cours...")

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)

Set_AnalogInput_Pin(board, PinUC)
Set_DigitalInput_Pin(board, PinButton)
Set_DigitalOutput_Pin(board, PinAlimC)

print("Connexion à l'Arduino établie - Appuyez sur Ctrl-C pour quitter.\n")

charge(board, PinAlimC, PinUC)

print("\nAppuyez sur le bouton poussoir pour commencer les mesures.\n")


# Boucle principale du programme

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        time.sleep(0.01)
        if ValButton == 1 and OldValButton == 0:
            State=1-State

        OldValButton = ValButton
```

```
        if State==1:
            if OldState == 0:
               print("\nDécharge du condensateur en cours.\n")
               print("Temps (S); Uc (V):")
               t0 = time.time()
               Digital_Write(board, PinAlimC, 0)
               OldState=1

            ValPinUC = Analog_Read(board, PinUC)
            UC = (ValPinUC/1023)*5.0
            dt = time.time()-t0
            print(round(dt,2),";",round(UC,2))

            time.sleep(0.1)

        else:
            if OldState == 1:
                print("\nFin des mesures.\n")
                charge(board, PinAlimC, PinUC)
                print("\nAppuyez sur le bouton poussoir pour commencer les mesures.\n")
                OldState = 0

    except KeyboardInterrupt:
        print("\nFin du programme.\n")
        decharge(board, PinAlimC, PinUC)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **"ConnectToArduino.py",** contenant les fonctions de connexion à l'Arduino via le protocole **"Firmata Express"**,

. Le module **"PymataExpressDef.Py"** regroupant toutes les fonctions utiles à l'utilisation de **"Pymata-express"** (fonction de déclaration des entrées et sorties, de lectures, d'écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause,

. La fonction **"charge"** pour charger le condensateur avant l'étude de la décharge,

. La fonction **"décharge"** pour décharger le condensateur à la fin du programme.

- Déclaration des constantes et variables :

. **PinUc = 0** (cst correspondant au n° de la broche A0 sur laquelle le condensateur est connecté)

. **PinAlimC = 2** (cst correspondant au n° de la broche alimentant le dipôle RC)

. **ValPinUC = 0** (variable pour stocker la valeur de la broche du condensateur)

. **Uc = 0.0** (variable pour stocker le résultat du calcul de la tension aux bornes du condensateur)

. **t0 = 0** (variable pour stocker le temps de début de décharge du condensateur)

. **dt = 0** (variable correspondant à la différence de temps en s entre les mesures de tension du condensateur et le temps de début de décharge)

. **PinButton = 12** (cst correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

. **ValButton = 0** (variable pour stocker la valeur de l'état logique de la broche du bouton poussoir)

. **OldValButton = 0** (variable pour stocker la valeur précédente de l'état logique de la broche du bouton poussoir)

. **State=0** (variable correspondant à l'action à effectuer)

. **OldState = 0** (variable pour stocker la valeur précédente de la variable **State**)

. **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d'ouverture du port COM sélectionné et connexion à l'Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l'Arduino est réussie :

- Déclaration de la broche du condensateur en entrée analogique :

Set_AnalogInput_Pin(board, PinUC)

- Déclaration de la broche du bouton poussoir en entrée numérique :

Set_DigitalInput_Pin(board, PinButton)

- Déclaration de la broche alimentant le dipôle RC en sortie numérique :

Set_DigitalOutput_Pin(board, PinAlimC)

- Charge du condensateur :

--> Appel de la fonction **"charge"** : **charge(board, PinAlimC, PinUC)**

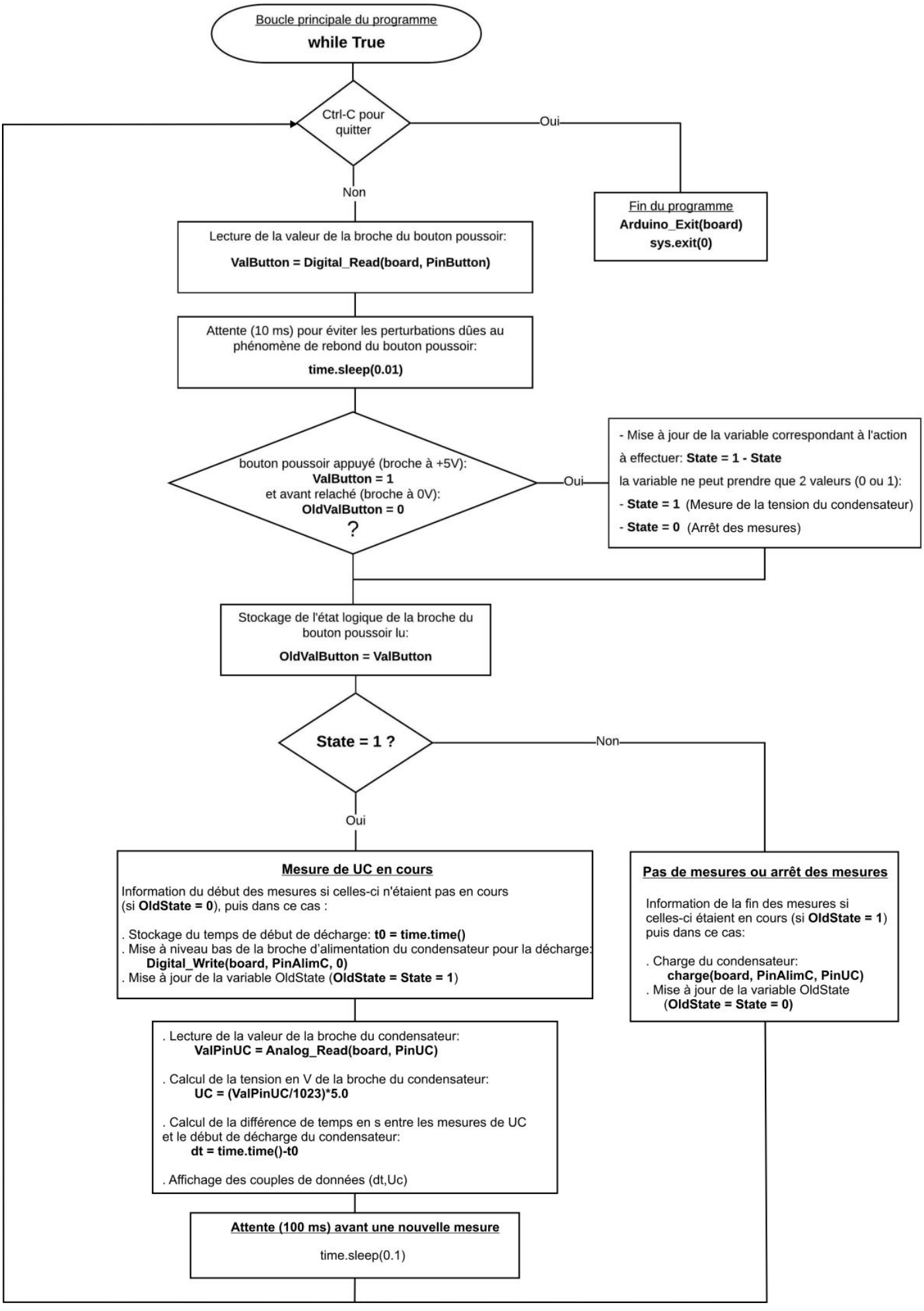
. Mise à niveau haut de la broche d'alimentation du dipôle RC:

Digital_Write(board, pinalim, 1)

. Attente de la charge complète du condensateur:

```
while Analog_Read(board, pinU) < 1023:
    time.sleep(0.2)
```

- Boucle principale du programme (boucle "while True") :


Boucle principale du programme
while True
Ctrl-C pour quitter
Oui
Non
Fin du programme
Arduino_Exit(board)
sys.exit(0)
Lecture de la valeur de la broche du bouton poussoir:
ValButton = Digital_Read(board, PinButton)
Attente (10 ms) pour éviter les perturbations dûes au phénomène de rebond du bouton poussoir:
time.sleep(0.01)
bouton poussoir appuyé (broche à +5V):
ValButton = 1
et avant relaché (broche à 0V):
OldValButton = 0
?
Oui
- Mise à jour de la variable correspondant à l'action à effectuer: State = 1 - State
la variable ne peut prendre que 2 valeurs (0 ou 1):
- State = 1 (Mesure de la tension du condensateur)
- State = 0 (Arrêt des mesures)
Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Non
Oui
Mesure de UC en cours
Information du début des mesures si celles-ci n'étaient pas en cours (si OldState = 0), puis dans ce cas :
. Stockage du temps de début de décharge: t0 = time.time()
. Mise à niveau bas de la broche d'alimentation du condensateur pour la décharge:
Digital_Write(board, PinAlimC, 0)
. Mise à jour de la variable OldState (OldState = State = 1)
Pas de mesures ou arrêt des mesures
Information de la fin des mesures si celles-ci étaient en cours (si OldState = 1) puis dans ce cas:
. Charge du condensateur:
charge(board, PinAlimC, PinUC)
. Mise à jour de la variable OldState
(OldState = State = 0)
. Lecture de la valeur de la broche du condensateur:
ValPinUC = Analog_Read(board, PinUC)
. Calcul de la tension en V de la broche du condensateur:
UC = (ValPinUC/1023)*5.0
. Calcul de la différence de temps en s entre les mesures de UC et le début de décharge du condensateur:
dt = time.time()-t0
. Affichage des couples de données (dt,Uc)
Attente (100 ms) avant une nouvelle mesure
time.sleep(0.1)

. Résultats dans la fenêtre Python Shell

```
Connexion à l'Arduino établie - Appuyez sur Ctrl-C pour quitter.

Charge du condensateur.
Condensateur chargé.

Appuyez sur le bouton poussoir pour commencer les mesures.


Décharge du condensateur en cours.

Temps (S); Uc (V):
0.0 ; 5.0
0.15 ; 4.41
0.28 ; 3.81
0.42 ; 3.42
0.55 ; 2.95
0.69 ; 2.61
0.85 ; 2.25
1.0 ; 1.95
1.14 ; 1.72
1.28 ; 1.49
1.42 ; 1.31
1.55 ; 1.16
1.69 ; 1.02
1.84 ; 0.89
1.98 ; 0.77

Fin des mesures.

Charge du condensateur.
Condensateur chargé.

Appuyez sur le bouton poussoir pour commencer les mesures.


Fin du programme.

Décharge du condensateur.
Condensateur déchargé.
>>>
```

. Programme en langage Arduino (”Projet7\Activity2\INO\Activity2.ino”)

Activity2

```
// Déclaration des constantes et variables

const int PinUC = 0;
const int PinButton = 12;
const int PinAlimC = 2;

int ValPinUC = 0;
float UC = 0.0;
unsigned long t0;
float dt;

int ValButton = 0;
int OldValButton = 0;
int State = 0;
int OldState = 0;


// Déclaration de fonctions

void charge(){
  digitalWrite(PinAlimC, HIGH);
  Serial.println("Charge du condensateur");
  while (analogRead(PinUC) < 1023) {
    delay(200);
    }
  Serial.println("Condensateur charge");
}


// Initialisation des entrées et sorties

void setup() {
  Serial.begin(9600);
  pinMode(PinButton, INPUT);
  pinMode(PinAlimC, OUTPUT);
  charge();
  Serial.println("Appuyez sur le bouton poussoir pour commencer les mesures.");
}


// Fonction principale en boucle

void loop() {
  ValButton = digitalRead(PinButton);
delay(10);

if ((ValButton == HIGH)&&(OldValButton == LOW))
  {
  State=1-State;
  }
OldValButton = ValButton;
```

```
if (State==1)
  {
  if (OldState == 0)
  {
   Serial.println("Decharge du condensateur en cours.");
   Serial.println("");
   Serial.println ("Temps (S);Uc (V):");
   t0 = micros();
   digitalWrite(PinAlimC, 0);
   OldState=1;
  }
  ValPinUC = analogRead(PinUC);
  UC = (ValPinUC/1023.0)*5.0;
  dt =  (micros() - t0)* 1e-6;
  Serial.print(dt,2);
  Serial.print(";");
  Serial.println(UC,2);

  delay(100);
  }
  else
  {
    if (OldState == 1){
    Serial.println("Fin des mesures.");
    charge();
    OldState = 0;}
  }
}
```

Déroulement du programme :

- Déclaration des constantes et variables :

. **const int PinUc = 0** (broche du condensateur : A0)
. const int PinButton = 12 (broche du bouton poussoir)
. **const int PinAlimC = 2** (broche d'alimentation du dipôle RC)

. **int ValPinUc = 0** (variable nombre entier valeur broche du condensateur)
. **float Uc = 0.0** (variable nombre décimal calcul tension Uc)
. unsigned long t0 (variable nombre entier long temps début charge)
. float dt (variable nombre décimal différence de temps entre les mesures de UC et le temps de début de décharge)

. int ValButton = 0 (variable nombre entier valeur broche bouton poussoir)
. int OldValButton = 0 (variable nombre entier ancienne valeur broche bouton poussoir)
. int State = 0 (variable nombre entier pour action à effectuer)
. int OldState = 0 (variable nombre entier pour action effectuée précédemment)

- Déclaration de fonctions :

→ Fonction permettant de charger le condensateur :

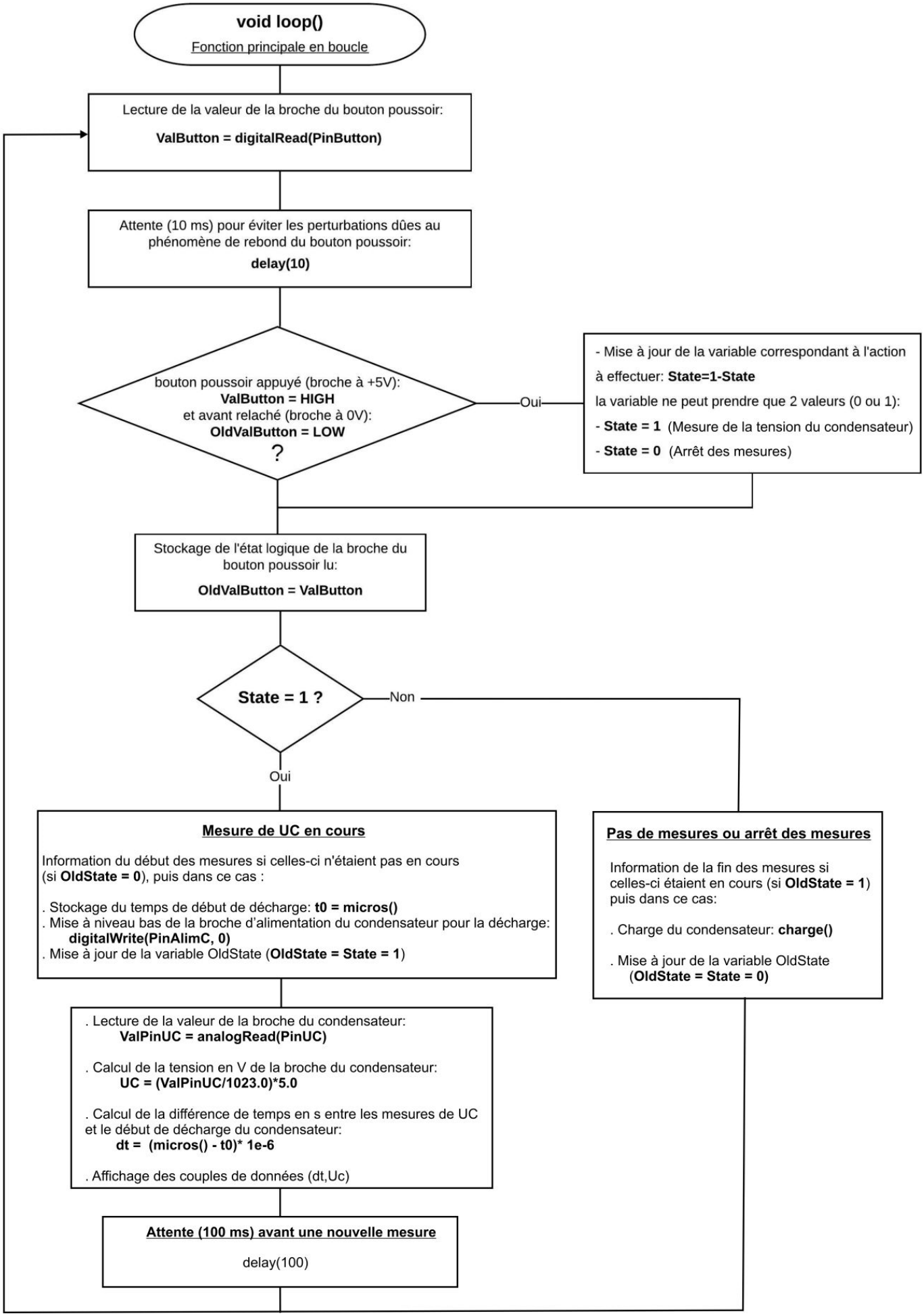
- Mise à niveau haut de la broche d'alimentation du dipôle RC :
  digitalWrite(PinAlimC, HIGH)

- Attente de la fin de la charge du condensateur :
  while (analogRead(PinUC) < 1023) ;
  delay(200) ;

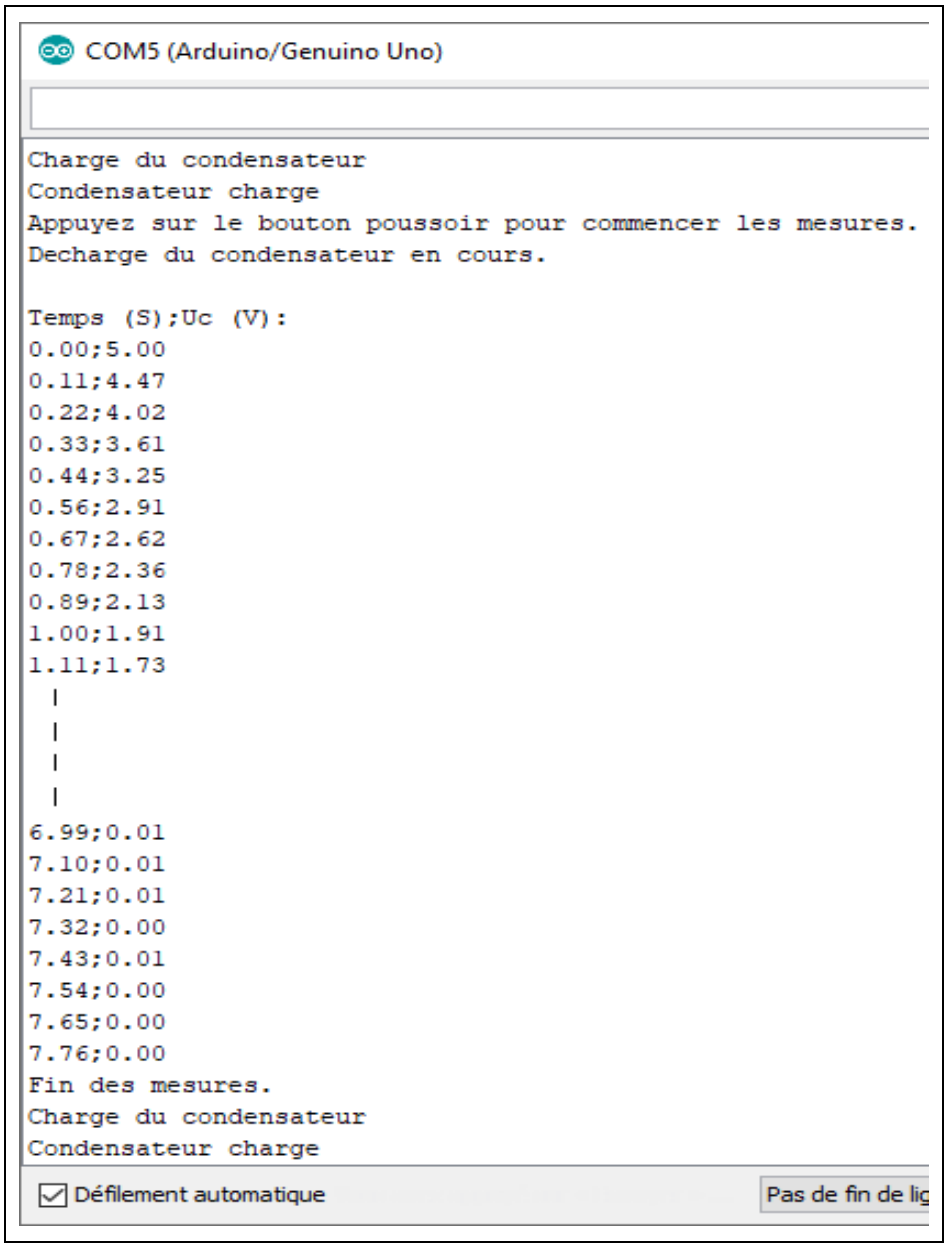
- Initialisation des entrées et sorties :

. Initialisation de la liaison série à un débit de 9600 bauds,
. Initialisation de la broche du bouton poussoir en entrée numérique,
. Initialisation de la broche d'alimentation du dipôle RC en sortie numérique,
. Charge du condensateur.

- Fonction principale en boucle :

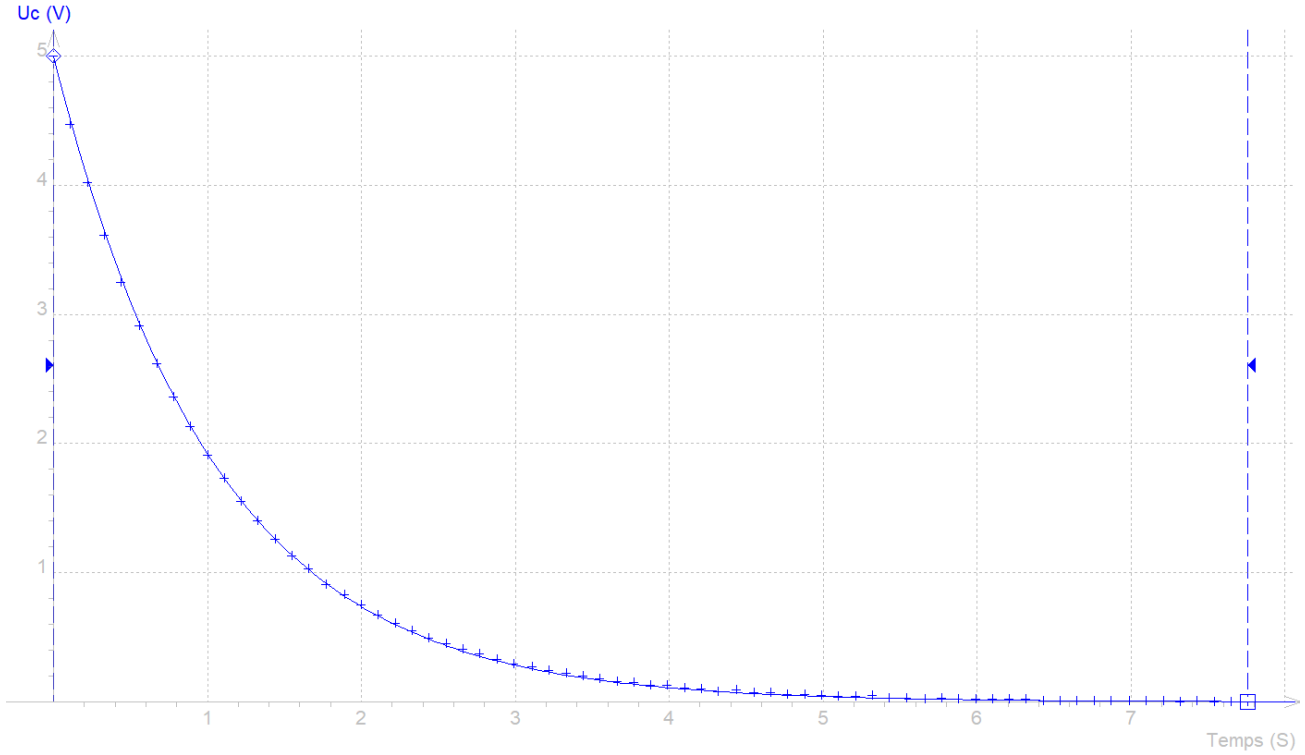

void loop()
Fonction principale en boucle
Lecture de la valeur de la broche du bouton poussoir:
ValButton = digitalRead(PinButton)
Attente (10 ms) pour éviter les perturbations dûes au phénomène de rebond du bouton poussoir:
delay(10)
bouton poussoir appuyé (broche à +5V):
ValButton = HIGH
et avant relaché (broche à 0V):
OldValButton = LOW
?
Oui
- Mise à jour de la variable correspondant à l'action à effectuer: State=1-State
la variable ne peut prendre que 2 valeurs (0 ou 1):
- State = 1 (Mesure de la tension du condensateur)
- State = 0 (Arrêt des mesures)
Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Non
Oui
Mesure de UC en cours
Information du début des mesures si celles-ci n'étaient pas en cours (si OldState = 0), puis dans ce cas :
. Stockage du temps de début de décharge: t0 = micros()
. Mise à niveau bas de la broche d'alimentation du condensateur pour la décharge: digitalWrite(PinAlimC, 0)
. Mise à jour de la variable OldState (OldState = State = 1)
Pas de mesures ou arrêt des mesures
Information de la fin des mesures si celles-ci étaient en cours (si OldState = 1) puis dans ce cas:
. Charge du condensateur: charge()
. Mise à jour de la variable OldState (OldState = State = 0)
. Lecture de la valeur de la broche du condensateur:
ValPinUC = analogRead(PinUC)
. Calcul de la tension en V de la broche du condensateur:
UC = (ValPinUC/1023.0)*5.0
. Calcul de la différence de temps en s entre les mesures de UC et le début de décharge du condensateur:
dt = (micros() - t0)* 1e-6
. Affichage des couples de données (dt,Uc)
Attente (100 ms) avant une nouvelle mesure
delay(100)

. **Résultats dans le moniteur série :**



. **Exploitation des mesures :**

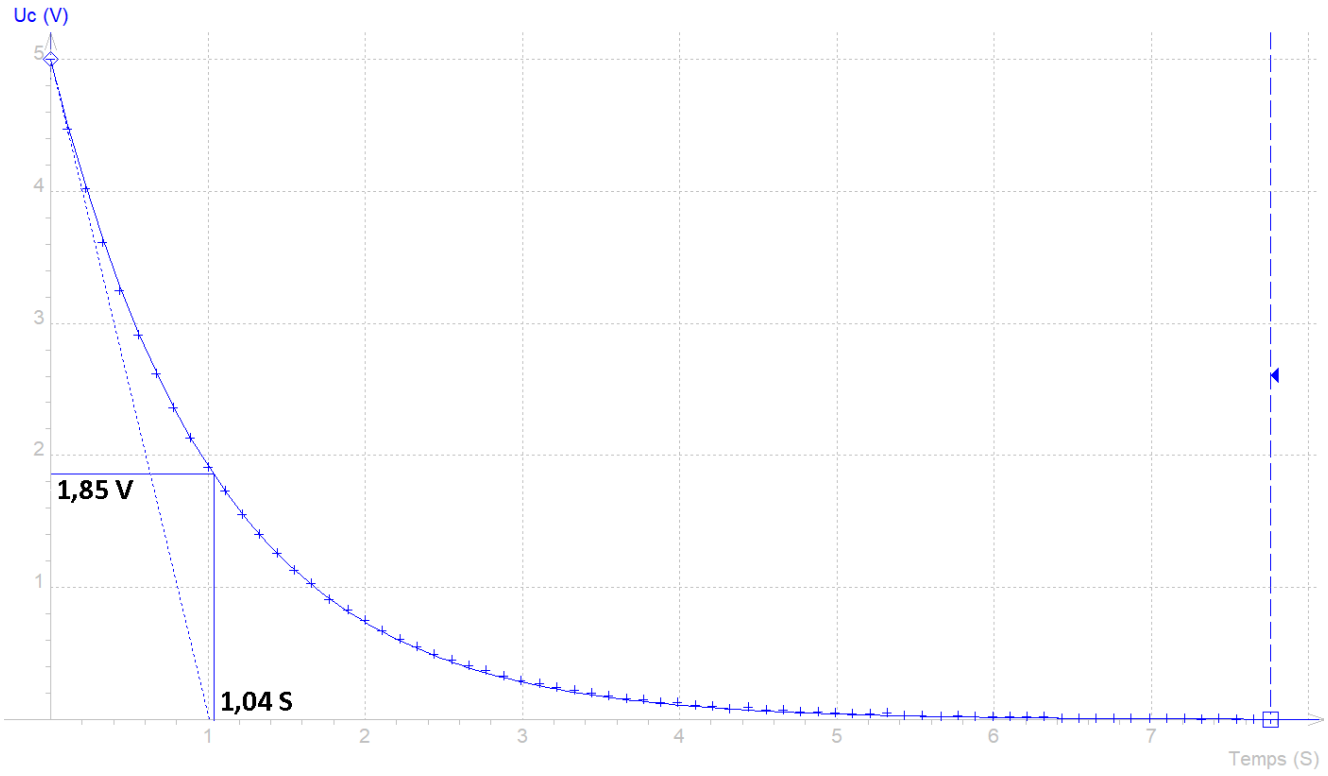
De même que précédemment, on utilise Regressi pour effectuer la modélisation de la représentation graphique de Uc=f(t) :



Expression du modèle	Résultats de la modélisation
Uc(Temps)=5*exp(-Temps/tau)	Ecart expérience-modèle 0,89 % sur Uc(Temps) Ecart quad. Uc=12,1 mV tau=1,045 ±0,003

Par la modélisation, la détermination de τ donne également une valeur très proche de la valeur théorique.

On peut également déterminer τ à l'aide de la tangente à l'origine ou par la mesure du temps pour lequel le condensateur est déchargé à 37 % (0,37 x 5 = 1,85 V):



- Activité 3 : Détermination de la capacité d'un condensateur par mesure d'une constante de temps

. Objectif

Dans cette activité, nous allons modifier le programme de l'activité 1 (étude de la charge d'un condensateur) de façon cette fois-ci à mesurer directement la constante de temps du circuit RC sans passer par le tracé de la caractéristique Uc=f(t).

On pourra alors déterminer la capacité d'un condensateur inconnu puisque :

$\tau = RC$ donc : $C = \tau / R$

. Descriptif de l'activité

Lors de la charge d'un condensateur, nous avons vu, qu'au bout d'un temps égal à la constante de temps τ, le condensateur était chargé à 63 % de la tension appliquée au dipôle RC (ici, à $t = \tau$: Uc= 0,63 x 5 = 3,15 V).

Avec le même circuit que les activités précédentes, pour déterminer la constante de temps, il suffit donc, par lecture de l'entrée analogique A0, de mesurer le temps que met cette entrée pour atteindre la valeur correspondant à la tension aux bornes du condensateur au temps τ, soit : **0,63 x 1023 = 644** (CAN 5 V = 1023).

Ainsi, après avoir déchargé le condensateur, la mesure de la tension aux bornes du condensateur U_c, lors de la charge, à l'aide de l'entrée analogique A0 est lancée, à t = 0 s, par un appui sur le bouton poussoir.

Quand la valeur de l'entrée analogique A0 souhaitée est atteinte, la constante de temps est alors calculée et affichée dans le moniteur série, puis le condensateur est déchargé.

Une nouvelle mesure est possible en appuyant de nouveau sur le bouton poussoir.

. Le programme

Voici le code de l'activité en Python et en langage Arduino :

. Programme en Python ("Projet7\Activity3\PY\Activity3.py")

```
# Importations des librairies et définition de fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time

def decharge(board, pinalim, pinU):
    Digital_Write(board, pinalim, 0);
    print("Décharge du condensateur.")
    while Analog_Read(board, pinU) > 0:
        time.sleep(0.2)
    print("Condensateur déchargé.")


# Déclaration des constantes et variables

PinUC = 0
PinButton = 12
PinAlimC = 2

ValPinUC = 0
t0 = 0
dt = 0

ValButton = 0
OldValButton = 0
State = 0


# Connexion à l'Arduino

print("\nConnexion à l'Arduino en cours...")

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)

Set_AnalogInput_Pin(board, PinUC)
Set_DigitalInput_Pin(board, PinButton)
Set_DigitalOutput_Pin(board, PinAlimC)


print("Connexion à l'Arduino établie - Appuyez sur Ctrl-C pour quitter.\n")

decharge(board, PinAlimC, PinUC)

print("\nAppuyez sur le bouton poussoir pour commencer les mesures.\n")


# Boucle principale du programme

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        time.sleep(0.01)
        if ValButton == 1 and OldValButton == 0:
            State=1-State

        OldValButton = ValButton;

        if State==1:
            print("\nCharge du condensateur en cours.\n")
            Digital_Write(board, PinAlimC, 1)
            t0 = time.time()
            while ValPinUC<644:
                ValPinUC = Analog_Read(board, PinUC)
```

```
            dt = time.time()-t0
            print("Tau = ",round(dt,2))
            print("\n\nFin des mesures.\n")

            decharge(board, PinAlimC, PinUC)
            State = 0
            ValPinUC = 0

            print("\nAppuyez sur le bouton poussoir pour commencer les mesures.\n")

    except KeyboardInterrupt:
        print("\nFin du programme.\n")
        decharge(board, PinAlimC, PinUC)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **"ConnectToArduino.py",** contenant les fonctions de connexion à l'Arduino via le protocole **"Firmata Express"**,

. Le module **"PymataExpressDef.Py"** regroupant toutes les fonctions utiles à l'utilisation de **"Pymata-express"** (fonction de déclaration des entrées et sorties, de lectures, d'écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause,

. La fonction **"décharge"** pour décharger le condensateur avant la charge et à la fin du programme.

- Déclaration des constantes et variables :

. **PinUc = 0** (cst correspondant au n° de la broche A0 sur laquelle le condensateur est connecté)

. **PinAlimC = 2** (cst correspondant au n° de la broche alimentant le dipôle RC)

. **ValPinUC = 0** (variable pour stocker la valeur de la broche du condensateur)

. **t0 = 0** (variable pour stocker le temps de début de charge du condensateur)

. **dt = 0** (variable correspondant à la différence de temps en s entre les mesures de tension du condensateur et le temps de début de charge)

. **PinButton = 12** (cst correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

. **ValButton = 0** (variable pour stocker la valeur de l'état logique de la broche du bouton poussoir)

. **OldValButton = 0** (variable pour stocker la valeur précédente de l'état logique de la broche du bouton poussoir)

. **State=0** (variable correspondant à l'action à effectuer)

. **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d'ouverture du port COM sélectionné et connexion à l'Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l'Arduino est réussie :

- Déclaration de la broche du condensateur en entrée analogique :

Set_AnalogInput_Pin(board, PinUC)

- Déclaration de la broche du bouton poussoir en entrée numérique :

Set_DigitalInput_Pin(board, PinButton)

- Déclaration de la broche alimentant le dipôle RC en sortie numérique :

Set_DigitalOutput_Pin(board, PinAlimC)

- Décharge du condensateur :

--> Appel de la fonction **"decharge"** : **decharge(board, PinAlimC, PinUC)**

. Mise à niveau bas de la broche d'alimentation du dipôle RC:

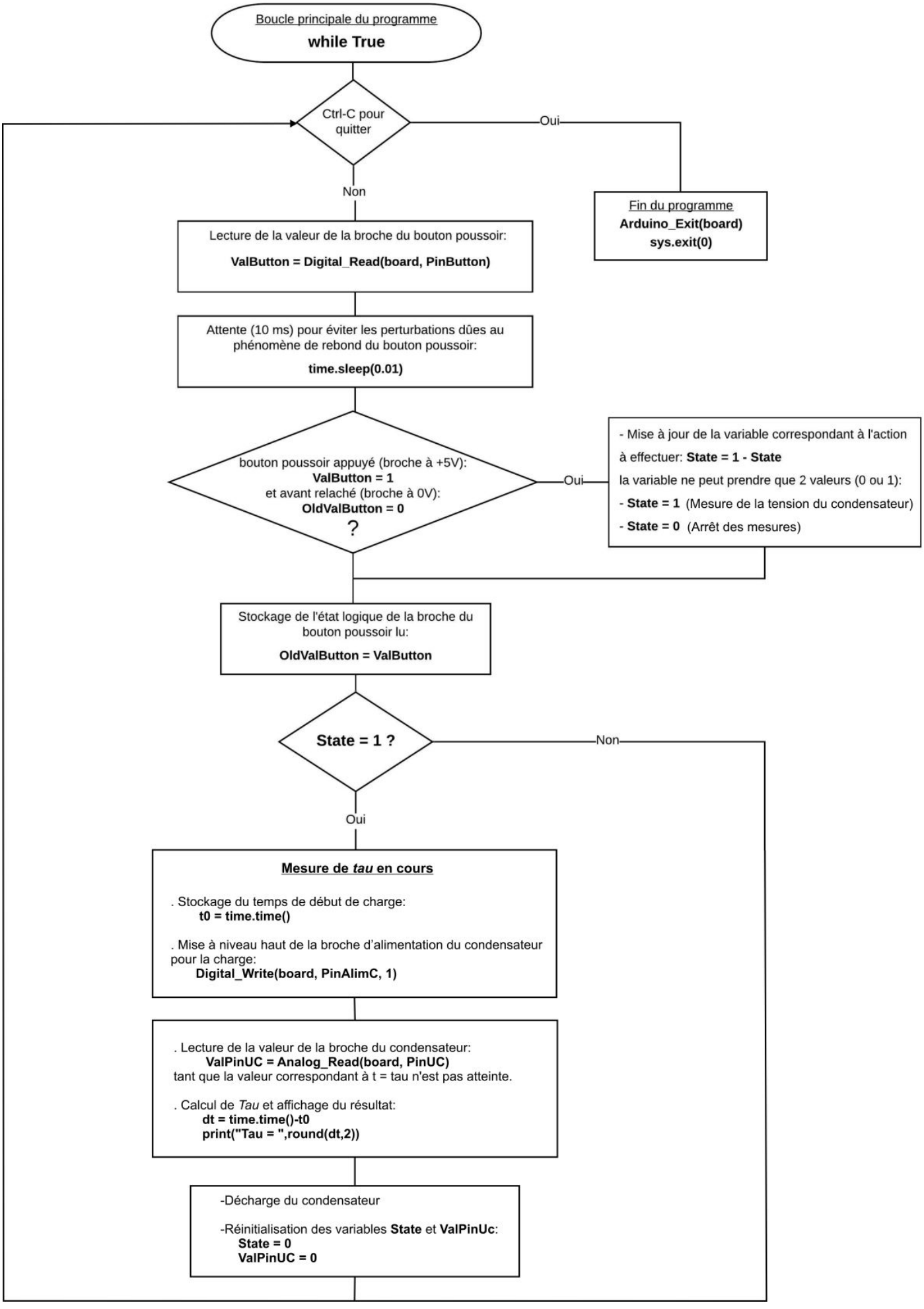
Digital_Write(board, pinalim, 0)

. Attente de la décharge complète du condensateur:

while Analog_Read(board, pinU) > 0:

time.sleep(0.2)

- Boucle principale du programme (boucle "while True") :


Boucle principale du programme
while True
Ctrl-C pour quitter
Oui
Non
Fin du programme
Arduino_Exit(board)
sys.exit(0)
Lecture de la valeur de la broche du bouton poussoir:
ValButton = Digital_Read(board, PinButton)
Attente (10 ms) pour éviter les perturbations dûes au phénomène de rebond du bouton poussoir:
time.sleep(0.01)
bouton poussoir appuyé (broche à +5V):
ValButton = 1
et avant relaché (broche à 0V):
OldValButton = 0
?
Oui
- Mise à jour de la variable correspondant à l'action à effectuer: State = 1 - State
la variable ne peut prendre que 2 valeurs (0 ou 1):
- State = 1 (Mesure de la tension du condensateur)
- State = 0 (Arrêt des mesures)
Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Non
Oui
Mesure de tau en cours
. Stockage du temps de début de charge:
t0 = time.time()
. Mise à niveau haut de la broche d'alimentation du condensateur pour la charge:
Digital_Write(board, PinAlimC, 1)
. Lecture de la valeur de la broche du condensateur:
ValPinUC = Analog_Read(board, PinUC)
tant que la valeur correspondant à t = tau n'est pas atteinte.
. Calcul de Tau et affichage du résultat:
dt = time.time()-t0
print("Tau = ",round(dt,2))
-Décharge du condensateur
-Réinitialisation des variables State et ValPinUc:
State = 0
ValPinUC = 0

. Résultats dans la fenêtre Python Shell

```
Connexion à l'Arduino établie - Appuyez sur Ctrl-C pour quitter.

Décharge du condensateur.
Condensateur déchargé.

Appuyez sur le bouton poussoir pour commencer les mesures.


Charge du condensateur en cours.

Tau =  1.05


Fin des mesures.

Décharge du condensateur.
Condensateur déchargé.

Appuyez sur le bouton poussoir pour commencer les mesures.


Fin du programme.

Décharge du condensateur.
Condensateur déchargé.
>>>
```

. Programme en langage Arduino (”Projet7\Activity3\INO\Activity3.ino”)

Activity3

```
// Déclaration des constantes et variables

const int PinUC = 0;
const int PinButton = 12;
const int PinAlimC = 2;

int ValPinUC = 0;
unsigned long t0;
float dt;

int ValButton = 0;
int OldValButton = 0;
int State = 0;


// Déclaration de fonctions

void decharge(){
  digitalWrite(PinAlimC, LOW);
  Serial.println("Decharge du condensateur.");
  while (analogRead(PinUC) > 0) {
    delay(200);
    }
  Serial.println("Condensateur decharge.");
}


// Initialisation des entrées et sorties

void setup() {
  Serial.begin(9600);
  pinMode(PinButton, INPUT);
  pinMode(PinAlimC, OUTPUT);
  decharge();
  Serial.println("Appuyez sur le bouton poussoir pour commencer les mesures.");
}


// Fonction principale en boucle

void loop() {
  ValButton = digitalRead(PinButton);
delay(10);

if ((ValButton == HIGH)&&(OldValButton == LOW))
  {
  State=1-State;
  }
OldValButton = ValButton;
```

```
if (State==1)
  {
  Serial.println("Charge du condensateur en cours.");
  Serial.println("");
  digitalWrite(PinAlimC, 1);
  t0 = micros();
  while (ValPinUC<644) {
    ValPinUC = analogRead(PinUC);
  }
  dt =  (micros() - t0)* 1e-6;
  Serial.print("Tau = ");
  Serial.println(dt,2);
  Serial.println("");
  Serial.println("Fin des mesures.");
  decharge();
  State = 0;
  ValPinUC = 0;
  Serial.println("Appuyez sur le bouton poussoir pour commencer les mesures.");
  }
}
```

Déroulement du programme :

- Déclaration des constantes et variables :

. **const int PinUc = 0** (broche du condensateur)
. const int PinButton = 12 (broche du bouton poussoir)
. **const int PinAlimC = 2** (broche d'alimentation du dipôle RC)

. **int ValPinUc = 0** (variable nombre entier valeur broche du condensateur)
. unsigned long t0 (variable nombre entier long temps début charge)
. float dt (variable nombre décimal correspondant à la différence de temps entre la mesure de la tension du condensateur à t=Tau et le début de la charge)

. int ValButton = 0 (variable nombre entier valeur broche bouton poussoir)
. int OldValButton = 0 (variable nombre entier ancienne valeur broche bouton poussoir)
. int State = 0 (variable nombre entier pour action à effectuer)

- Déclaration de fonctions :

→ Fonction permettant de décharger le condensateur :

- Mise à niveau bas de la broche d'alimentation du dipôle RC :
  digitalWrite(PinAlimC, LOW)

- Attente de la fin de la décharge du condensateur :
  while (analogRead(PinUC) > 0) ;
  delay(200)

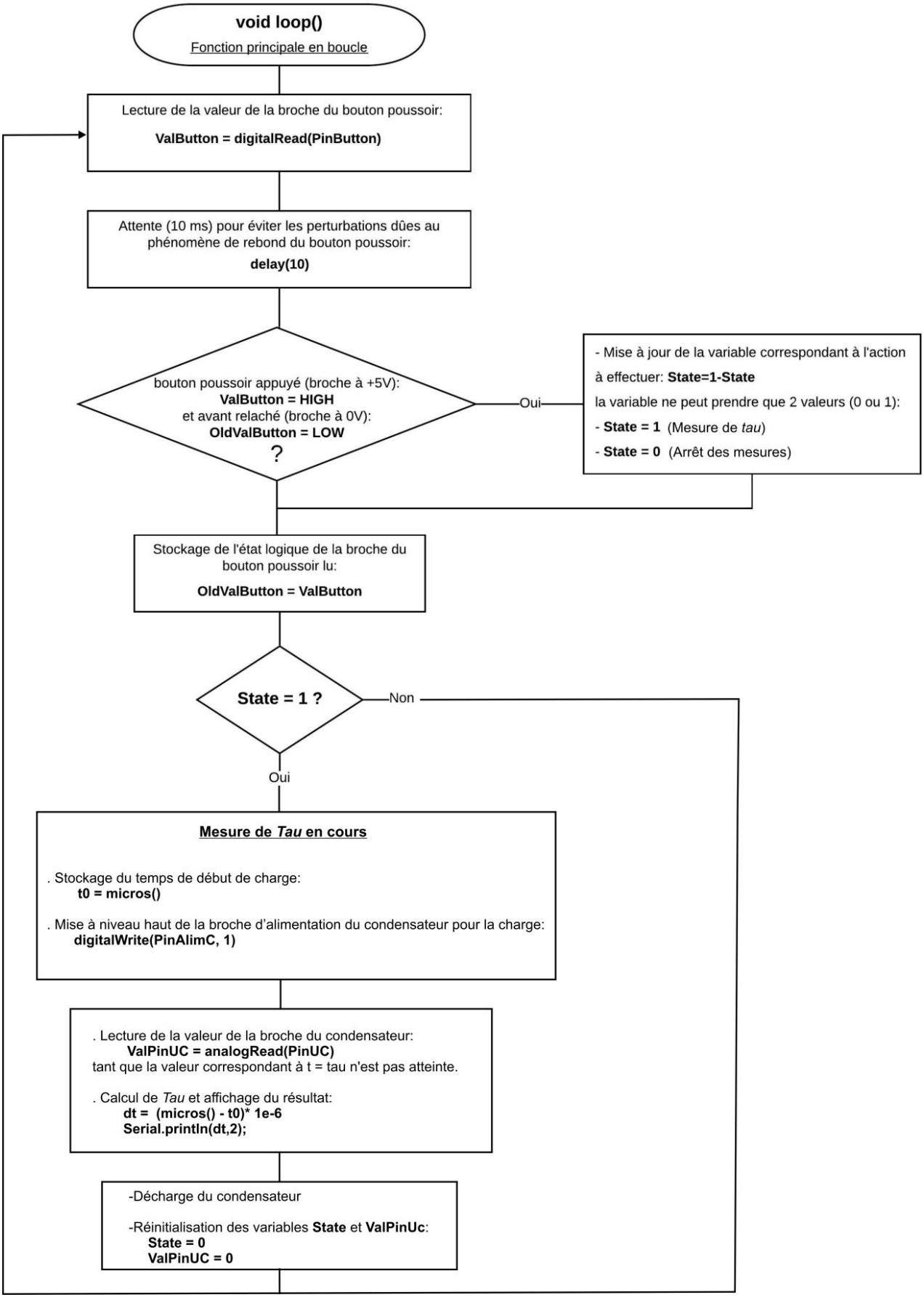
- Initialisation des entrées et sorties :

. Initialisation de la liaison série à un débit de 9600 bauds,
. Initialisation de la broche du bouton poussoir en entrée,

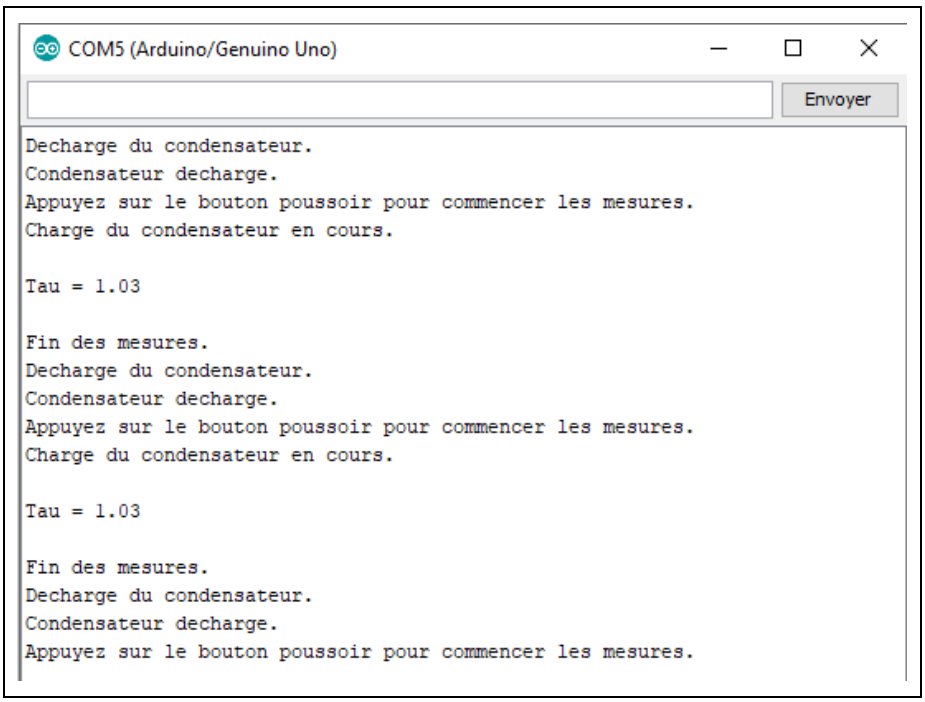
. Initialisation de la broche d'alimentation du dipôle RC en sortie,

. Décharge du condensateur.

- Fonction principale en boucle :



. Résultats dans le moniteur série :



. Exploitation des mesures

Les valeurs de τ mesurées par le code en Python et en langage Arduino sont conformes à celles obtenues lors de la première activité.

On peut alors déterminer la capacité du condensateur (valeur théorique de 100 µF) :

$C = \tau/R = 1{,}03 / 10.10^{3} = 1{,}03.10^{-4}$ F = 103 µF (105 µF avec le code en Python)