Étude de la charge d'un condensateur

⋆ ⋆ ⋆

Acquisition Arduino™, Traitement Python

1 Introduction

Ce sujet prend appui sur le programme de Terminale Spécialité (*Déterminer le temps caractéristique d'un dipôle RC à l'aide d'un microcontrôleur, ...*).

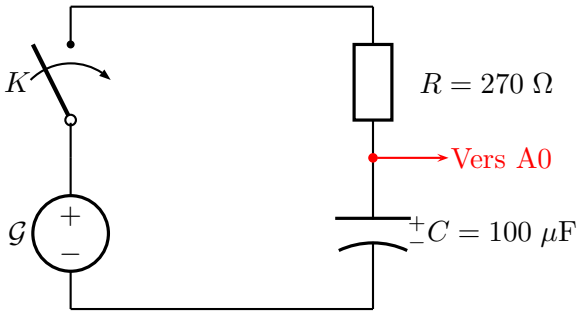
Il s'agit d'un "sujet profs" : l'objectif essentiel est ici de parcourir différents codes Arduino et Python permettant de traiter cette partie. Il est possible de faire communiquer entre eux les deux systèmes de programmation mais nous ne nous y intéresserons pas.

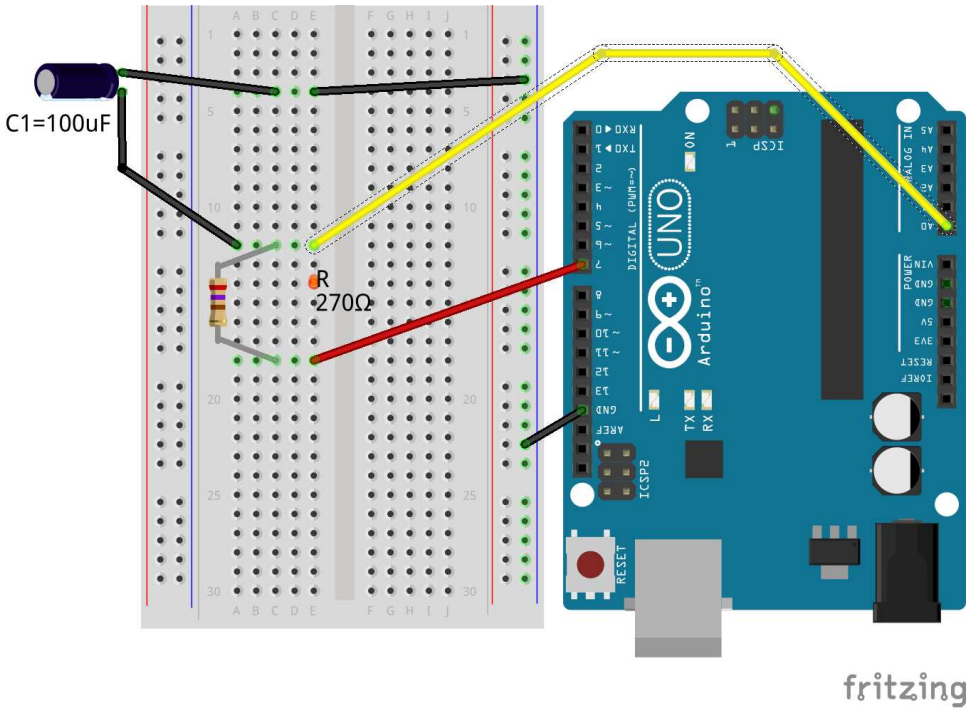
On considère un système avec une constante de temps de l'ordre de 10 à 100 ms qui correspond à des composants classiques ($R = 270\ \Omega$ et $C = 100\ \mu$F). Libre à vous de choisir d'autres valeurs.

NOTE : Les valeurs de capacité portées sur les condensateurs peuvent être très approximatives (exemples de condensateurs notés à 100 μF dont la capacité mesurée est autour de 80 μF).

Attention : lors de l'utilisation de condensateurs électrochimiques, il faut prendre garde à la polarisation : comme pour les LED, on relie la tige la plus longue du côté "+" de l'alimentation.

On souhaite suivre la charge ou la décharge du condensateur ; le montage du sujet est le suivant :





Commentaires :

— La charge ou décharge seront commandées par la mise en état haut ou bas de la broche 7 ; pour cela nous verrons une solution logicielle (par une commande Arduino). On peut imaginer une solution matérielle (on ferme le circuit par appui sur un bouton poussoir) mais le montage est alors plus évolué et pourrait gêner les élèves.

— C'est l'utilisateur qui choisit le moment de démarrage de la charge et des acquisitions. Pour cela, on passera par le **moniteur série** qui permet de voir les retours de la carte (par `Serial.print`) mais qui permet aussi d'envoyer des "ordres" à la carte.

— La variation de tension aux bornes du condensateur sera lue sur l'entrée analogique A0

⟶ En faisant relever par la carte les valeurs simultanées de temps et de charge du condensateur (lue sur A0), on peut ensuite tracer la courbe de charge par exemple en Python.

Le processus de traitement est le suivant :

Commande par le moniteur série ⟶ Acquisition par la carte Arduino ⟶ sortie des données (`temps;charge`) sur le moniteur série ⟶ Copier-coller vers un fichier type csv ⟶ traitement des données (tableur, Python)

2 Charge du condensateur : acquisition des données

2.1 Code Arduino™

Le code incomplet est fourni ici : `S2-charge1.ino`.
Répondez aux questions suivantes pour compléter le code :
BLOC SETUP :

1. Complétez le bloc SETUP en vous servant des commentaires fournis.
2. Pourquoi place-t-on la broche 7 à l'état bas dans ce bloc ? (ligne 12).
3. Le "baudrate" (vitesse de communication avec le port série a été choisi à 115 200 bits/s ; d'habitude, on utilise couramment un baudrate à 9600.
   Pourquoi avoir choisi ce "baudrate" pour cette acquisition ?
4. Dans l'optique d'une exploitation par Python ou par un tableur, quel est l'intérêt de la ligne 14 ?

BLOC LOOP :

5. Quelle est l'utilité de la condition ligne 20 ?
6. Pourquoi est-elle essentielle dans ce type d'acquisition ?
7. Finissez de remplir le bloc LOOP en vous servant des commentaires.
   REMARQUE : ligne 29, on choisit d'exprimer la charge du condensateur en pourcentage de la charge maximale.
   RAPPEL : le CAN de la carte réalise une conversion sur 10 bits soit 1024 valeurs de 0 à 1023. Sans réglage particulier, la valeur 1023 correspond à 5 Volts.

Lien vers la correction du code : `S2-correc-charge1.ino`

8. Compilez votre code ; ouvrez le moniteur série. Pensez à régler le "baudrate" à 115200 sur le moniteur série.
   Après l'affichage du "titre", la carte attend que vous rentriez quelque chose pour démarrer l'acquisition.
   Tapez une lettre dans la ligne du haut puis validez, l'acquisition démarre.

2.2 Généralisation du code Arduino

Une fois connues les caractéristiques de charge du condensateur (constante de temps τ, temps approximatif pour charger ou décharger le condensateur $\gtrapprox 5 \cdot \tau$), on peut modifier le programme précédent pour arriver à une seconde version :

`S2-charge2.ino`

⟹ Observez ce code ; expliquez l'intérêt des lignes modifiées (spécifiées par MODIF en commentaires).

3 Exploitation par Python

Pour relancer une acquisition, il suffit d'appuyer sur le bouton RESET de la carte.

Une fois les données reçues par le moniteur série, copiez-coller les dans un fichier que vous pourrez appeler `donnees1.csv`.

lien vers un fichier de données : `S2-donnees1.csv`

ÉNONCÉ "CONFIRMÉS"

On souhaite représenter l'évolution de la charge en fonction du temps.

Le programme Python ci-joint `S2-exploitation1.py` traite les données du fichier `S2-donnees1.csv`.

Complétez ce programme pour placer sur un graphe (T, Requiv) en bleu l'ensemble des points expérimentaux et en pointillés rouges la courbe de charge théorique ($q(t) = q_{\max} \times \left(1 - \mathrm{e}^{-t/\tau}\right)$)

Donnez un nom aux axes, un titre et une légende au graphe.

ÉNONCÉ "DÉBUTANTS"

On souhaite représenter l'évolution de la charge en fonction du temps.

Le programme Python ci-joint `S2-correc-exploitation1.py` traite les données du fichier `S2-donnees1.csv`.

Commentez toutes les lignes de ce programme qui place sur un graphe (temps, Charge) en bleu l'ensemble des points expérimentaux et trace aussi la courbe théorique en traits pointillés rouge.

4 Discussions

4.1 Importance du "baudrate"

Comme dit plus haut, le débit de communication sur le port série a été fixé à 115200 bits par seconde au lieu du classique 9600 utilisé fréquemment.

Afin de voir l'influence de ce baudrate, vous pouvez relancer une acquisition et une exploitation en fixant le baudrate à 9600.

Conclusions.

4.2 Détermination de la constante de temps, méthodes lycée

— Une fois le graphe obtenu, on peut déterminer la constante de temps par exemple en se déplaçant sur le graphe et en visualisant les coordonnées (en bas à droite de la fenêtre)

— On peut par exemple tracer une droite horizontale à 63% , puis placer la souris à l'intersection entre nos points expérimentaux et cette droite pour déterminer τ.
$\Longrightarrow$ À vous de tracer cette droite sur votre graphe.

solution 1 : `plt.plot([0, 50000], [63, 63], "--g")`

solution 2 : `plt.plot(temps, [63]*len(temps), "--g")`

Ces commandes tracent une ligne à 63% du maximum (si il a été posé à 100) sur tout le graphe ; on peut alors à la souris déterminer la coordonnée temporelle du point d'intersection entre cette droite et nos données.

— On peut de même faire tracer la tangente à l'origine et déterminer l'abscisse du point de la tangente d'ordonnée `ChargeMax`.

$\Longrightarrow$ Observez et intégrez le `bout de code ci-joint` à votre programme ; observez son effet.

En modifiant à la main (ou en créant une boucle pour plusieurs valeurs) la variable `tau_inconnue_us`, on peut approximer la constante de temps du circuit.

5 Décharge du condensateur

5.1 Code Arduino™

$\Longrightarrow$ Modifiez à la marge le programme `S2-charge2.ino` ou `S2-correc-charge1.ino` pour réaliser l'acquisition d'une décharge de condensateur.

lien vers une correction possible : `S2-decharge1.ino`

Enregistrez les données dans un fichier `S2-donnees2.csv`.

lien vers un fichier de données : `S2-donnees2.csv`

$\Longrightarrow$ Comment analysez-vous le fait que la charge maximale du condensateur ne démarre pas à 100% en début d'acquisition ?

5.1.1 Exploitation

Servez-vous du code Python précédent pour tracer les points expérimentaux ainsi que la courbe théorique de décharge.

$\Longrightarrow$ Une fois le graphe obtenu, on peut déterminer la constante de temps par exemple en se déplaçant sur le graphe et en visualisant les coordonnées.

lien vers un programme Python correctif : `S2-correc-exploitation2.py`