



PROJET TUTEURE MCPC

2022-2023



IUT DE LILLE

2022-2023

Calligaro Bastien / Cache Josse / Leroy Thibaut / Delcour Adrien

Sommaire :

Introduction	p2
I. Fonctionnement de la batterie.....	p3 - p10
1°) Principe de la batterie	p3 - p4
2°) État de charge théorique.....	p4 - p8
3°) Principe de la chronopotentiométrie.....	p8 - p9
4°) Acquisition de données.....	p10
II. Métrologie.....	p11 - p21
1°) Protocole et études des spectres d'absorbance des solutions Ferri et Ferro.....	p11 - p15
2°) Évolution de ϵ et détermination de son incertitude.....	p15 - p16
3°) Exemple d'essai au potentiomètre.....	p16 - p20
4°) Etalonnage de notre capteur de température.....	p20 - p21
III. Capteurs et logiciels.....	p22 - p33
1°) Détermination des outils nécessaires.....	p22 - p26
2°) Programmation	p27 - p33
IV. Conclusion.....	p34
V. Bibliographie.....	p35

Introduction :

Dans ce projet, notre objectif est de suivre le fonctionnement d'une batterie de type *red ox* au cours du temps, notamment en étudiant son état de charge. Les détails seront donnés plus loin ci-dessous.

Le projet se décompose en plusieurs étapes distinctes qui nous ont chacune aidé à étudier cette batterie.

Dans un premier temps, nous étudions l'expérimentation de la batterie.

C'est-à-dire comment faire fonctionner cette dernière, quels sont ces composants, ... Également quelles solutions allons-nous faire circuler dans la batterie en sachant que se devront se produire des réactions d'oxydo-réduction afin d'assurer un bon fonctionnement.

Dans un second temps, nous nous concentrons sur la partie métrologique du projet. La métrologie tient une place importante dans notre projet. Elle nous permet d'exposer nos données avec les incertitudes les plus faibles possibles en répétant les mesures. Ici, nous l'avons utilisée afin d'obtenir une incertitude minimale sur nos coefficients d'absorption molaire mesurés avec le spectromètre et déduits de la loi de Beer-Lambert.

Ensuite, nous allons parler du codage python crée afin de faire fonctionner la batterie à distance mais également de pouvoir récolter et traiter les données d'état de charge de la batterie.

Enfin, nous présenterons les résultats obtenus.

Dans ce projet nous avons chacun notre rôle même si certaines tâches sont effectuées en groupe, cela permet que ce projet soit organisé et qu'il se déroule dans les meilleurs délais, les voici :

- ⇒ Thibaut Leroy : Référent du groupe, manipulateur, théorie, bibliographie
- ⇒ Josse Cache : Métrologue, manipulateur, théorie
- ⇒ Bastien Calligaro : Manipulateur, théorie
- ⇒ Adrien Delcour : Programmeur, manipulateur

I. Fonctionnement de la batterie

1°) Principe de la batterie

La batterie mis à disposition est une batterie redox flow, son utilisation est schématisée ci-dessous (Figure 1) :

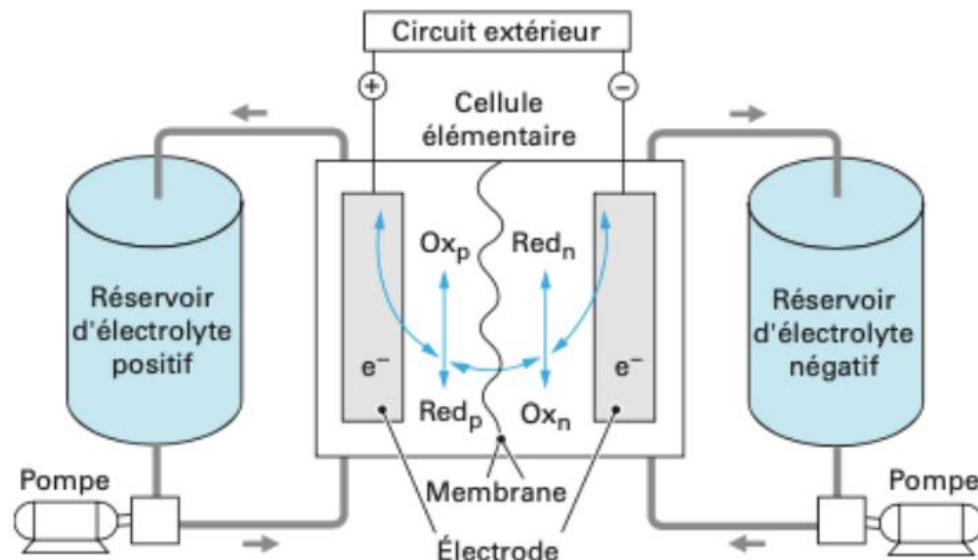


Figure 1 : Schéma d'une batterie

Les générateurs électrochimiques dans ce type de batterie se caractérisent par le fait que leurs matériaux actifs et produits de réaction sont en solution.

Les réservoirs d'électrolyte sont les réservoirs qui vont contenir les couples red/ox utilisés afin de faire fonctionner la batterie. Dans notre cas, nous utiliserons deux fois le même couple **Fe³⁺/Fe²⁺**.

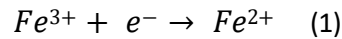
Les batteries red ox sont des batteries qui fonctionnent avec des réactions d'oxydo-réduction (donc de transfert d'électron).

Il existe à l'heure actuelle deux types de batterie industrielles :

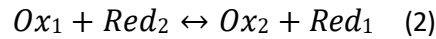
- ⇒ Batterie au vanadium
- ⇒ Batterie PEM

Les batteries au vanadium ont une haute densité massique d'énergie. Toutefois, le vanadium est un métal peu abondant mais aussi un métal toxique, ce qui peut limiter son utilisation dans l'industrie. Enfin, les batteries PEM sont des batteries fonctionnant par échange de proton et d'électron.

L'équation suivante (1) nous aide à déterminer l'état de charge (SOC) de la batterie à un instant donné. L'écriture de cette demi-équation est donc :

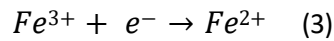


Théoriquement, l'état de charge d'une batterie nous ait donné par l'équation suivante :



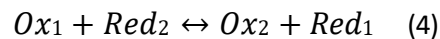
2°) État de charge théorique

L'écriture de la demi-équation nous donne donc :



Cette équation nous aide à déterminer l'état de charge (SOC) de la batterie à un instant donné.

Théoriquement, l'état de charge d'une batterie nous ai donné par l'équation suivante :



L'état de charge théorique de la batterie se calcule avec les concentrations de Fe^{2+} et de Fe^{3+} :

$$SOC = \frac{[Fe^{3+}]}{[Fe^{3+}] + [Fe^{2+}]} \quad (5)$$

Cette équation nous permet de déduire que l'état de charge sera de 100% lorsque :

$$[Fe^{2+}] = 0 \quad (6)$$

Mais aussi qu'il sera de 0% quand :

$$[Fe^{3+}] = 0 \quad (7)$$

Présentation du matériel :

- Batterie à flux RedOx :



Figure 2 : Vue d'ensemble de la batterie à flux RedOx

Après avoir démonté la batterie, on détaille les différentes parties qui la composent. Celles-ci sont assemblées en série (Figure 3) du pôle + au pôle - :

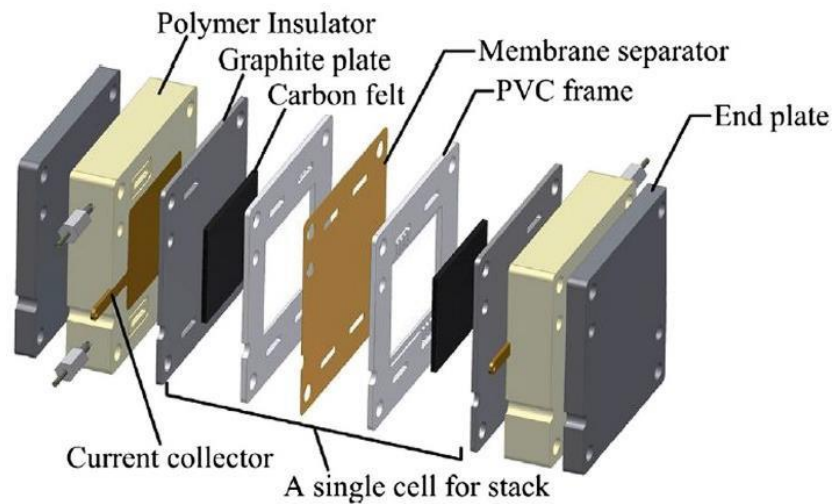


Figure 3 : Composants d'une batterie à flux redox au vanadium

(A review of all-vanadium redox flow battery durability:
Degradation mechanisms and mitigation strategies)

Composants de la batterie à flux redox utilisée

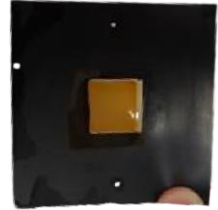
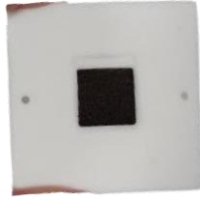


Figure 4 : Armatures conductrices	Figure 5 : Collecteur feutre de carbone encadré d'une membrane en PVC	Figure 6 : « Flow » en graphite	Figure 7 : Membrane échangeuse d'ions composé d'un polymère (nafion**)
---	---	---	--

nafion** : Voir ci-dessous

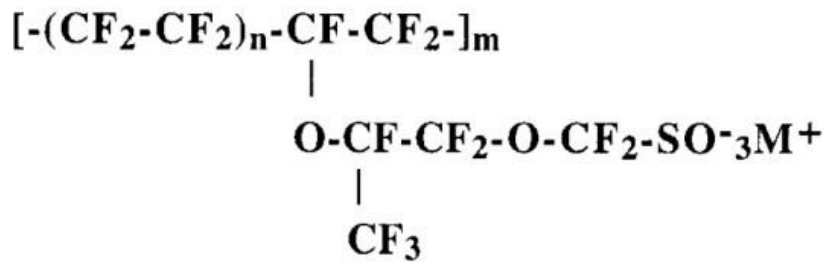


Figure 8 : Structure chimique du nafion**

**Le nafion confère à la membrane ses propriétés de conduction protonique. Celui-ci est utilisé pour les piles à combustible échangeuse de protons appelée PEMFC pour « proton exchange membrane fuel cell ». Le nombre de transports des ions H^+ en son sein est égal à 1. La température de fonctionnement est inférieure à 100°C .

Pendant le fonctionnement de la pile, un phénomène appelé crossover peut arriver. Il est causé par les espèces chimiques qui peuvent migrer à travers la membrane de nafion. Une infime partie de l'hydrogène est susceptible de diffuser de l'anode vers la cathode et de réagir chimiquement avec l'oxygène pour former de l'eau. Cette réaction engendre un gaspillage de carburant qui peut s'avérer non négligeable. De plus, comme la réaction est exothermique, des points chauds peuvent apparaître aux endroits où le crossover est important et provoquer une dégradation thermique.

A fraction d'eau élevée, (>0.5) le volume d'eau étant plus élevé que celui du polymère, la membrane peut être dissoute et la structure peut s'inverser. (Phénomène de microstructure vue en RDM)

- Pompe rotative multicanaux :



Cette pompe péristaltique permet d'empiler 2 à 4 têtes de pompes sur le même entraînement et peut pomper le liquide les 2 sens de rotation.

Grâce aux différentes vitesses de rotation proposées, on peut jouer sur le débit de la pompe avec la commande numérique.

Figure 9 : Image de la pompe péristaltique utilisée

Pour voir clairement ce que nous décrivions auparavant voici un schéma théorique du dispositif (Figure 10) et une photo du dispositif expérimentale (Figure 11) :

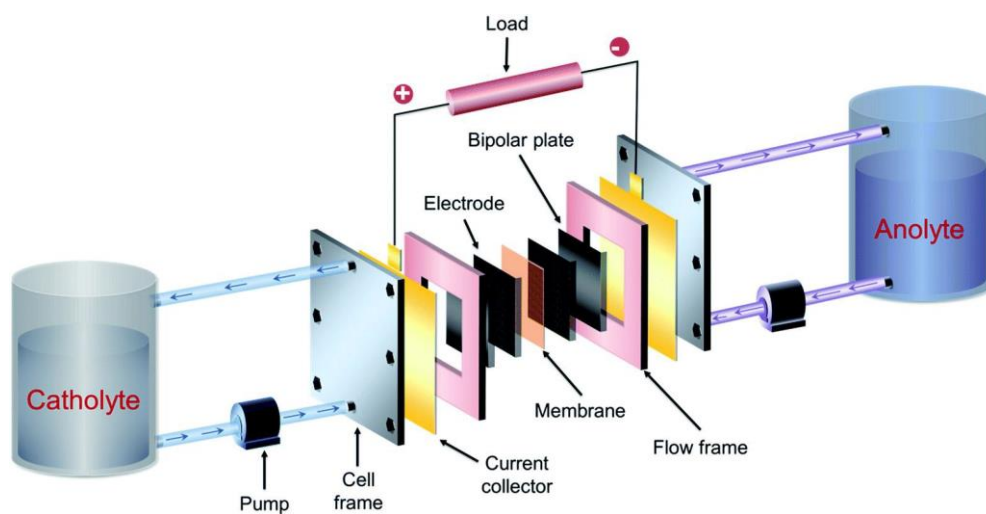


Figure 10 : Schéma théorique du dispositif

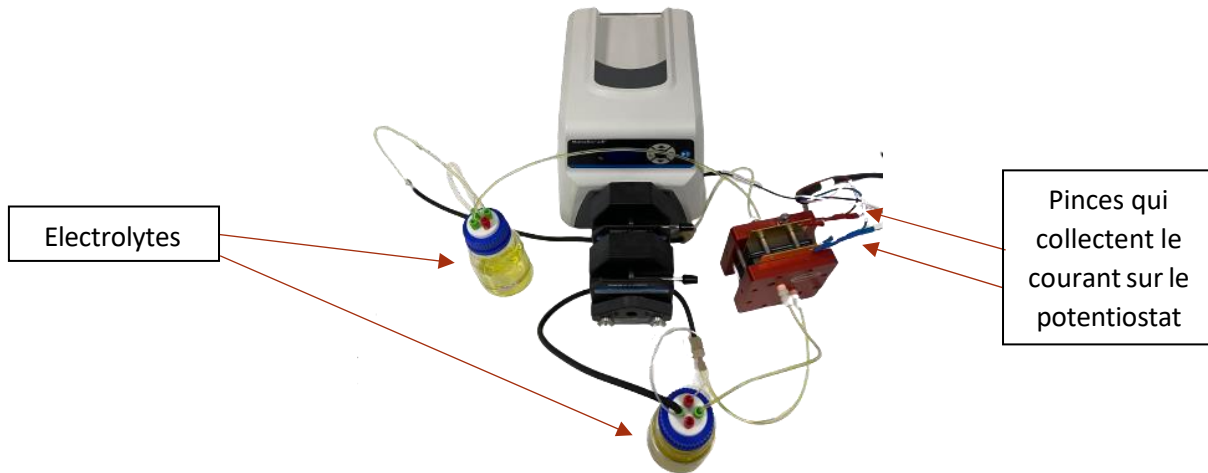


Figure 11 : Schéma du dispositif expérimental

3°) Principe de la chronopotentiométrie (CP)

Le principe de la CP est d'appliquer un courant au système et de mesurer l'évolution de la tension de celui-ci en sortie.

Les plaques sont traversées par le même courant (loi de Kirchhoff) et la tension du Stack est égale à la somme des tensions de chaque cellule. (Dans notre cas système modèle composé d'une seule cellule)

La caractérisation de PEM (Proton exchange membrane) par CP consiste à imposer un courant alternativement positif et négatif au système afin de faire évoluer le potentiel du système dans la gamme souhaitée (Figure 12).

Durant l'étape où le courant est positif, la tension de l'électrode de travail va augmenter plus ou moins rapidement selon les processus électrochimiques mis en jeu. Et inversement lorsque le courant est négatif, la tension de travail diminue. Plusieurs cycles sont effectués afin de s'assurer de la reproductibilité des résultats (Figure 12)

La tension maximale qu'atteint la cellule sera appelée "tension de retour ou d'arrivée". La tension minimale à la fin du cycle k ou au début du cycle k+1 sera appelée "tension de départ". En pratique, on prendra soin d'éviter que l'électrode de travail n'atteigne des potentiels trop élevés (< 1 V) afin d'éviter son altération, la corrosion et l'électrolyse de l'eau.

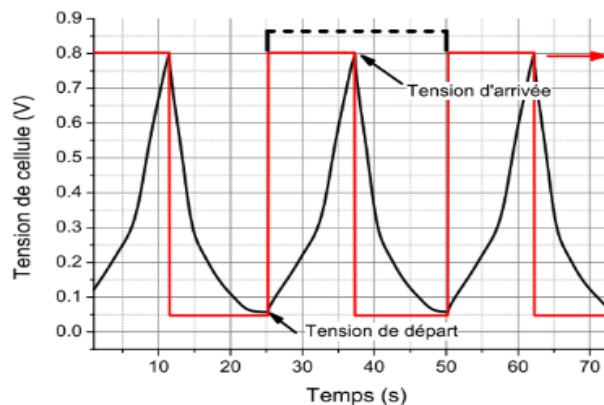


Figure 12 : Créneaux de courants appliqués à une monocellule en tension

Si $i > 0$ on est dans une phase de charge, on observe une oxydation à l'anode qui se trouve au niveau de la polarisation positive et dans ce cas on observe la réduction à la cathode qui se trouve au niveau de la polarisation négative. On se trouve donc dans un mode électrolyseur on convertit l'énergie électrique en énergie chimique.

Lorsque $i < 0$ on est dans une phase de décharge, les polarisations sont inversées par rapport aux réactions d'oxydoréduction. On se trouve donc dans un mode de pile ou de générateur on convertit l'énergie chimique en énergie électrique.

Ces réactions sont grandement simplifiées par le modèle symétrique (modèle) de la batterie.

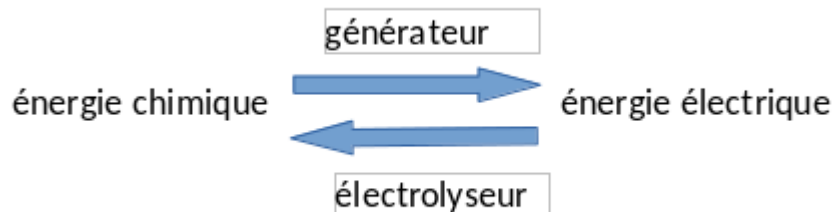


Figure 13 : Schéma des réactions dans une pile

Lorsque l'on impose un courant de décharge d'intensité constante I_d à un générateur, on constate que la différence de potentiel aux bornes décroît en fonction du temps. L'allure de la courbe de décharge dépend du type de générateur. Certains systèmes donnent lieu à des courbes très plates [piles Li (Figure 14)].

En effet, on remarque que lorsqu'on impose un courant de décharge d'intensité constante à un générateur, on constate que la différence de potentiel chute plus ou moins rapidement en fonction des processus électrochimiques mis en jeu au cours du temps.

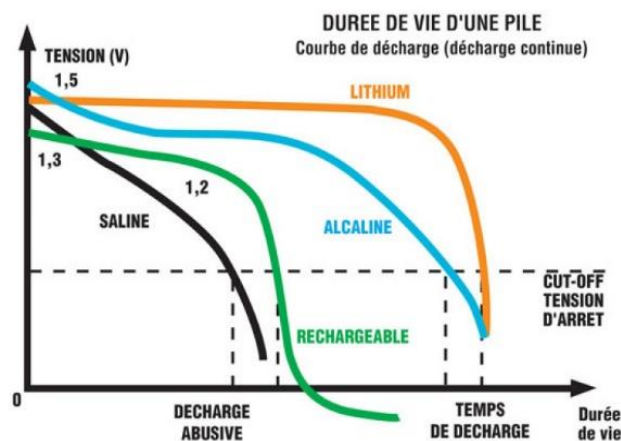


Figure 14 : Courbes de décharges $U=f(V)$ de différentes piles
(source Pile 4,5 V type 3R12 Saline – Pierron)

4°) Acquisition de données

On fait fonctionner la pompe à l'aide du logiciel "Ec_lab". On fixe le courant à faire circuler dans la pompe selon la charge Q qui a pour formule :

$$Q = i \times t = n_e \times F \times nFe \quad (8)$$

- ⇒ Si $i > 0$ alors oxydation au pôle +
- ⇒ Si $i < 0$ alors réduction pôle +

Avec :

- ⇒ n_e le nombre d'électron (ici = 1)
- ⇒ F la constante de Faraday (= 96 488)
- ⇒ nFe la quantité de matière de Fe en mole, se calculera grâce au volume et à la concentration molaire

Le volume des électrolytes est de 100 mL et la concentration molaire des solutions est fixée à 0,1 mol.L⁻¹.

On en déduit :

$$Q = 1 \times 96488 \times (0,1 \times 0,1) = 964,88 \text{ C} \quad \Rightarrow \quad 268.02 \text{ mAh} \quad (9)$$

Pour les manipulations on arrondira à 275 mAh.

En ce qui concerne les potentiels, on ne dépassera pas 1 V pour le potentiel car possibilité de créer des réactions parasites tel que l'électrolyse de l'eau à 1.5V par exemple. Cette limite fixée à 1V nous permet de ne pas créer des produits tel que l'oxygène ou l'hydrogène.

II. Métrologie

1°) Protocole et études des spectres d'absorbance des solutions Ferri et Ferro

Protocole :

On désire étudier les spectres de solutions hexacyanoferrate de potassium (Ferri) et de Ferrocyanure de potassium (Ferro) afin de déterminer les longueurs d'ondes maximales. Grâce à cette mesure de λ , on mesure l'absorbance ce qui va nous permettre de déduire le coefficient d'extinction molaire ϵ .

Les mesures d'absorbances étant faites à différentes T°C de la solution, on pourra tracer l'évolution de ϵ en fonction de la T°C. L'équation de la courbe nous permettra de déterminer le coefficient d'extinction molaire pour une température donnée lors des acquisitions de batterie. Etant donné que l'absorbance sera mesurée grâce à un capteur piloté par un code python et la concentration est connue. Le but final étant donc de déterminer notre état de charge expérimental.

On prend pour cela des températures allant de 10°C à 45 par pas de 5, 3 mesures par pic caractéristique.

En ce qui concerne les mesures à 10°C, nous utilisons un gaz inerte (argon) afin d'éviter les réactions parasites (dans les CNTP) tels que la condensation (choc thermique important qui provoque de l'humidité) qui est aussi à l'origine de l'oxydation avec le dioxygène du Ferro par exemple. Ce qui a provoqué un changement de teinte, c'est pourquoi les solutions étaient de nouveau préparées toutes les 2 semaines environ.

Dans un premier temps, on lance le logiciel « spectrum » qui va nous permettre de piloter le Varian Cary 100 qui est le spectromètre UV-Visible ainsi que le Varian Cary dual cell, l'accessoire nous permettant de faire varier la température du système.



Les paramètres de mesures (onglet Setup) sont les suivants :

- Gamme de mesure de 700 à 250 nm (onglet Cary)
- Réglage de la température (onglet accessoires 1) en cochant « Automatic Temperature Setting » et 1 x 1 peltier. Puis on saisit la température dans les onglets « Rear » et « front »

On valide les paramètres et on lance l'acquisition. Les pics sont reportés grâce à l'option de « tracking » où l'on observe l'absorbance maximale.

Etudes des spectres d'absorbance des solutions Ferri et Ferro

Afin de tracer notre variation du ε en fonction de la $T^{\circ}\text{C}$, on a besoin de mesurer l'absorbance de nos solutions de Ferri et de Ferro à une concentration définie. Ce qui nous permet d'utiliser la loi de Beer-Lambert :

$$A = \varepsilon \times l \times C \quad (10)$$

Avec :

- ⇒ $l = 1\text{cm}$ (largeur des cuvettes en quartz) qui correspond au trajet optique
 - ⇒ C la concentration molaire en mol.L^{-1}
 - ⇒ A l'absorbance sans unité
 - ⇒ ε le coefficient d'extinction molaire $\text{L.mol}^{-1}.\text{cm}^{-1}$
-
- Les solutions utilisées pour les spectres sont à 1 mmol.L^{-1} pour ne pas avoir de signal saturé (groupe de l'année dernière) en raison de la largeur de la cuve qui est de 1cm.
 - Pour cela, on prépare les solutions mères de ferrocyanure de potassium et ferricyanure de potassium (degré d'oxydation II et III). On fixe leur concentration à 0.05 mol.L^{-1} afin de réaliser les solutions filles à 1 mmol.L^{-1} dans des fioles de 50 mL .
 - On cherche les masses à prélever pour nos dissolutions car les deux composés sont des solides (poudres).

On utilise la formule suivante :

$$m = C \times V \times M \quad (11)$$

Avec :

- ⇒ $m = n \times M$
- ⇒ $n = C \times V$

Avec les termes :

- ⇒ m : la masse en grammes
- ⇒ C : la concentration en mol.L^{-1}
- ⇒ V : le volume en mL
- ⇒ M : la masse molaire en g.mol^{-1}

Ici nous avons :

$$m(\text{Ferro}) = 0,05 \times 0,05 \times 422,39 \approx 1,056 \text{ g}$$

et

$$m(\text{Ferri}) = 0,05 \times 0,05 \times 329 \approx 0,823 \text{ g}$$

On utilise la formule de dilution :

$$C_f \times V_m \rightarrow V_m = 1 \text{ mL} \quad (12)$$

On prélève donc 1 mL de solution mère puis on complète avec de l'eau distillée jusqu'au trait de jauge pour obtenir les solutions filles.

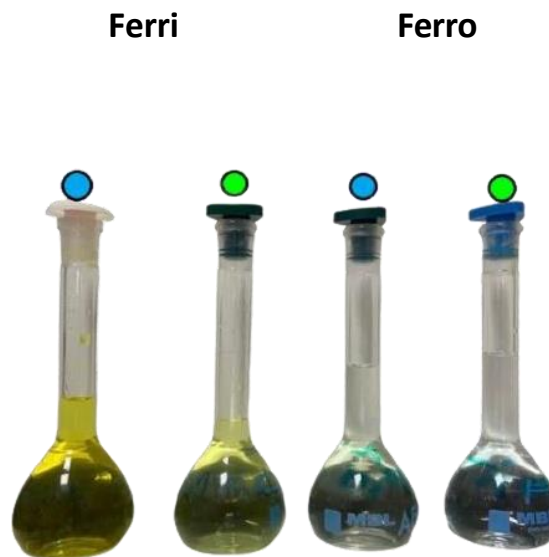


Figure 16 : Fiole de solution de Ferri et de Ferro

Légende :

- $V_{\text{fille}} = 0,001 \text{ mol/L}$
- $V_{\text{mère}} = 0,05 \text{ mol/L}$

On obtient pour les 2 molécules un spectre caractéristique avec des pics représentatifs (on précise que l'on choisit un spectre par espèce toutes T°C confondues pour analyser la forme générale) :

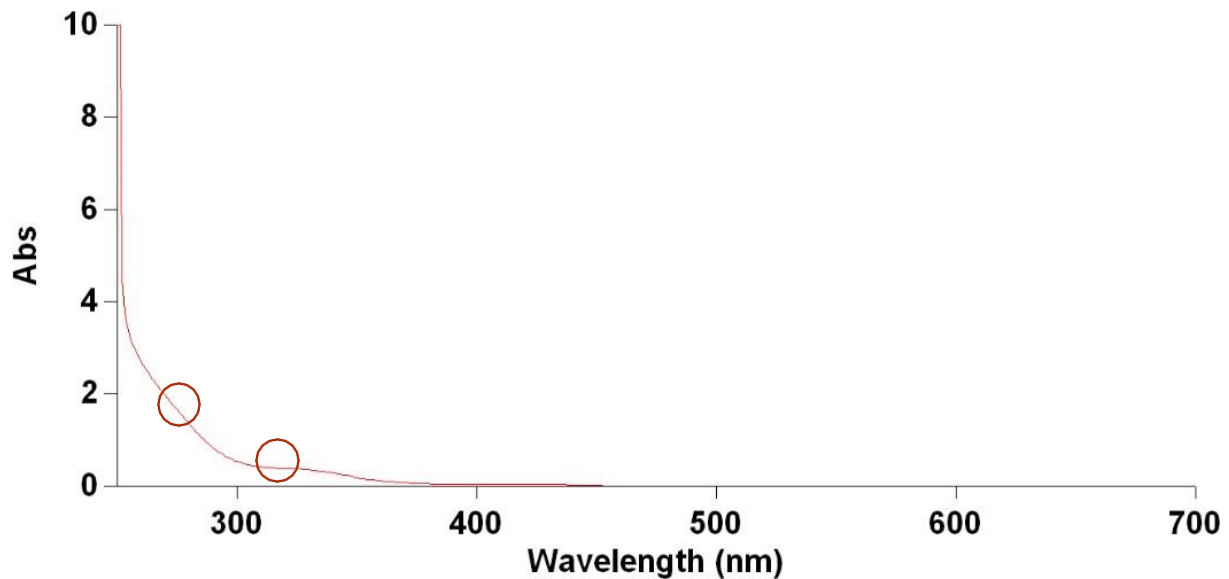


Figure 17 : Spectre du Ferro (espèce ionique étudiée Fe^{2+})

En ce qui concerne le Fe^{2+} , nous avons 2 pics caractéristiques situés à environ 286 nm (saturé) et à 320 nm situés dans l'UV. Le pic à 286 nm étant trop saturé, il ne sera pas retenu pour la suite de l'étude sur l'état de charge SOC. De manière générale, les valeurs de l'absorbance pour une longueur d'onde inférieure à 300 nm sont « erronées » en raison de la saturation à 1 mmol.L^{-1} , il faudrait une cuve avec une largeur inférieure à 1cm par exemple.

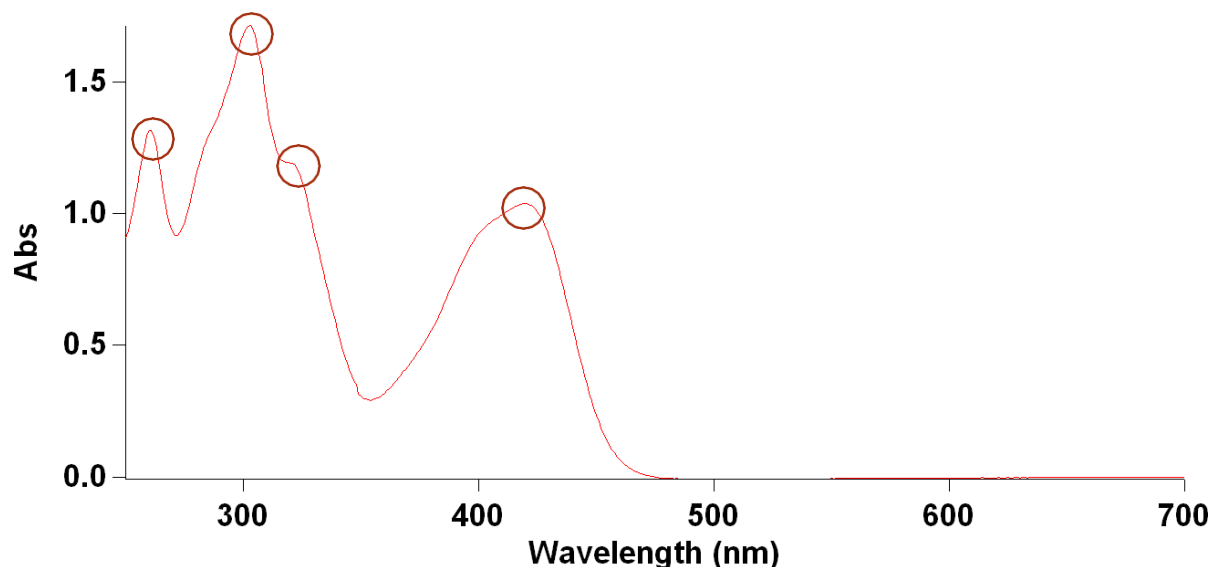


Figure 18 : Spectre du Ferri (espèce ionique étudiée Fe^{3+})

Pour le Fe^{3+} , nous avons 4 pics caractéristiques situés à environ 261, puis 303, 320 dans l'UV et 420 nm dans le visible.

Pour les manipulations avec le spectromètre fibré, nous garderons la longueur d'onde à 420 nm car à cette valeur de λ $A_{Fe^{2+}} \approx 0$. Et seule l'absorbance de Fe^{3+} est prise en compte, ce qui nous permet de maximiser l'état de charge de la batterie et ses performances.

Calcul du SOC :

$$SOC = \frac{[Fe^{3+}]}{C_0} \quad \text{avec} \quad C_0 = [Fe^{2+}] + [Fe^{3+}] \quad (13)$$

(On rappelle que la longueur l du trajet optique n'est pas présente dans les relations car égale à 1cm)

Hypothèse de C_0 constant :

En l'absence de migration des espèces et de n'importe quel phénomène pouvant faire varier la concentration des solutions tel que le « crossover ». (Voir page 6)**

A 420 nm :

$$A_{420} = [Fe^{3+}] \times \epsilon_{420}^{Fe^{3+}} \quad \text{car} \quad A_{420}^{Fe^{2+}} \approx 0 \quad (14)$$

donc

$$\epsilon_{420}^{Fe^{3+}} \gg \epsilon_{420}^{Fe^{2+}} \quad (15)$$

On en déduit :

$$SOC = \frac{A_{420}}{\epsilon_{420}^{Fe^{3+}} + C_0} \quad (16)$$

Par exemple pour un SOC de 100 %, l'absorbance de Fe^{2+} étant faible, la concentration peut donc être augmenté tandis que pour Fe^{3+} , la concentration sera plus faible car l'espèce absorbe beaucoup.

2°) Évolution de ϵ et détermination de son incertitude

Le but est de réduire un maximum l'incertitude liée à la répétabilité. (On rappelle que on relève l'absorbance à chaque pic caractéristique et on calcul ϵ grâce à la concentration.)

On effectue un total de 3 mesures pour chaque pic à chaque $T^\circ C$ ce qui nous permet de calculer la moyenne et l'écart-type dans un premier temps.

Puis on calcule les IC à 95 % afin de rajouter sur notre courbe $\epsilon=f(T^\circ C)$ les barres d'erreurs.

$$IC = \frac{s}{\sqrt{n}} * t(k, \gamma) \quad (17)$$

Le coefficient t se lit sur la table de Student avec $\gamma = 0.025$ et $k=n-1$ c'est-à-dire que t vaut 4.303 pour 3 mesures.

k	γ										
	0.25	0.20	0.15	0.10	0.05	0.025	0.010	0.005	0.0025	0.0010	0.0005
1	1.000	1.376	1.963	3.078	6.314	12.71	31.82	63.66	127.3	318.3	636.6
2	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	14.09	22.33	31.60
3	0.765	0.978	1.250	1.638	2.353	3.182	4.541	5.841	7.453	10.21	12.92
4	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	5.598	7.173	8.610
5	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	4.773	5.893	6.869
6	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	4.317	5.208	5.959
7	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	4.029	4.785	5.408
8	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	3.833	4.501	5.041
9	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	3.690	4.297	4.781
10	0.700	0.879	1.093	1.372	1.812	2.228	2.764	3.169	3.581	4.144	4.587
11	0.697	0.876	1.088	1.363	1.796	2.201	2.718	3.106	3.497	4.025	4.437
12	0.695	0.873	1.083	1.356	1.782	2.179	2.681	3.055	3.428	3.930	4.318
13	0.694	0.870	1.079	1.350	1.771	2.160	2.650	3.012	3.372	3.852	4.221
14	0.692	0.868	1.076	1.345	1.761	2.145	2.624	2.977	3.326	3.787	4.140
15	0.691	0.866	1.074	1.341	1.753	2.131	2.602	2.947	3.286	3.733	4.073
16	0.690	0.865	1.071	1.337	1.746	2.120	2.583	2.921	3.252	3.686	4.015
17	0.689	0.863	1.069	1.333	1.740	2.110	2.567	2.898	3.222	3.646	3.965
18	0.688	0.862	1.067	1.330	1.734	2.101	2.552	2.878	3.197	3.610	3.922
19	0.688	0.861	1.066	1.328	1.729	2.093	2.539	2.861	3.174	3.579	3.883
20	0.687	0.860	1.064	1.325	1.725	2.086	2.528	2.845	3.153	3.552	3.850
21	0.686	0.859	1.063	1.323	1.721	2.080	2.518	2.831	3.135	3.527	3.819
22	0.686	0.858	1.061	1.321	1.717	2.074	2.508	2.819	3.119	3.505	3.792
23	0.685	0.858	1.060	1.319	1.714	2.069	2.500	2.807	3.104	3.485	3.767
24	0.685	0.857	1.059	1.318	1.711	2.064	2.492	2.797	3.091	3.467	3.745
25	0.684	0.856	1.058	1.316	1.708	2.060	2.485	2.787	3.078	3.450	3.725
26	0.684	0.856	1.058	1.315	1.706	2.056	2.479	2.779	3.067	3.435	3.707
27	0.684	0.855	1.057	1.314	1.703	2.052	2.473	2.771	3.057	3.421	3.690
28	0.683	0.855	1.056	1.313	1.701	2.048	2.467	2.763	3.047	3.408	3.674
29	0.683	0.854	1.055	1.311	1.699	2.045	2.462	2.756	3.038	3.396	3.659
30	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.030	3.385	3.646
40	0.681	0.851	1.050	1.303	1.684	2.021	2.423	2.704	2.971	3.307	3.551
50	0.679	0.849	1.047	1.299	1.676	2.009	2.403	2.678	2.937	3.261	3.496
60	0.679	0.848	1.045	1.296	1.671	2.000	2.390	2.660	2.915	3.232	3.460
80	0.678	0.846	1.043	1.292	1.664	1.990	2.374	2.639	2.887	3.195	3.416
100	0.677	0.845	1.042	1.290	1.660	1.984	2.364	2.626	2.871	3.174	3.390
120	0.677	0.845	1.041	1.289	1.658	1.980	2.358	2.617	2.860	3.160	3.373

Figure 19 : Table du test de Student

3) Exemple d'essai au potentiomètre

Nous avons effectué une première série de mesure au potentiomètre.

Pour cette première série de mesure nous avons comme paramètre : une charge de 275 mAh (comme indiqué plus tôt) et nous avons décidé de lancer notre première série de mesure à une intensité de 100mA correspondant un peu près à 36% de charge pour la batterie.

Aussi, on a rappelé plus tôt que $Q=i \times t$, ainsi le temps correspondant à un cycle charge + décharge de la batterie est de :

$$t = \frac{Q}{i} = \frac{275}{100} = 2,75 \text{ heures} = 2\text{h } 45\text{min} = 9910 \text{ secondes} \quad (18)$$

Ainsi après avoir réalisé les mesures nous en obtenons le graphique suivant :

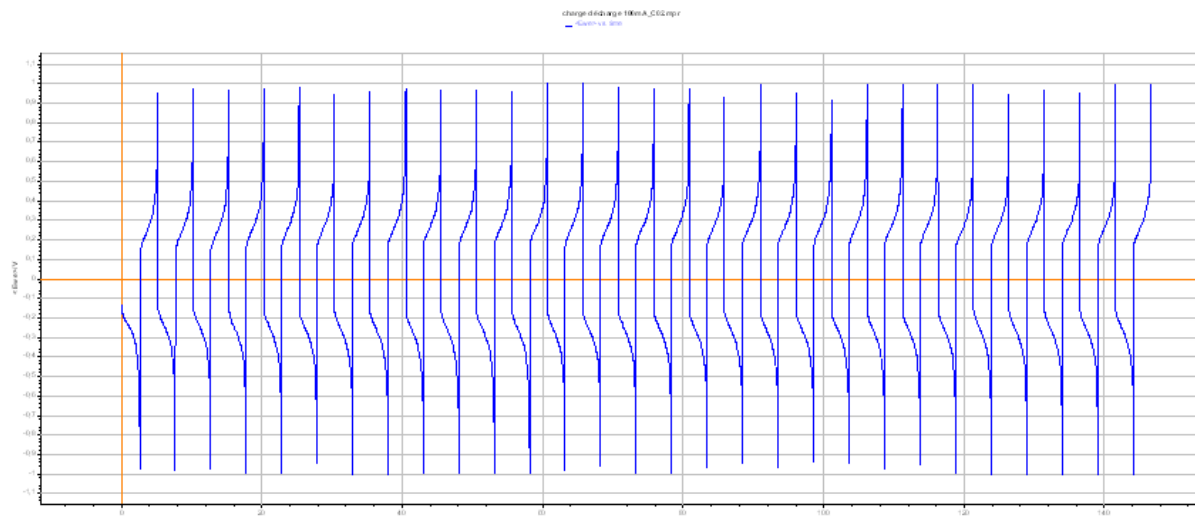


Figure 20 : Première acquisition de charge et décharge de la batterie

Nous pouvons observer comme demandé une succession de cycle de charge et décharge sans interruption. Nous n'observons aucun changement brutal de variation de tension dans le temps. Ainsi, l'absorbance reste aussi constante dans le temps. (Formule 16)

Après avoir pointé et sorti les valeurs des maximums et des minimums, nous pouvons étudier le rendement de la batterie dans cette première série de mesure.

Nous pouvons dans un premier temps comparer le temps de charge + décharge théorique à celui observé expérimentalement.

Sachant que notre temps de charge + décharge théorique est de 9910 secondes, ce temps est celui indiqué par l'ordinateur suite aux paramètres que nous lui avons indiqué et précisé précédemment (voir ci-dessous) :

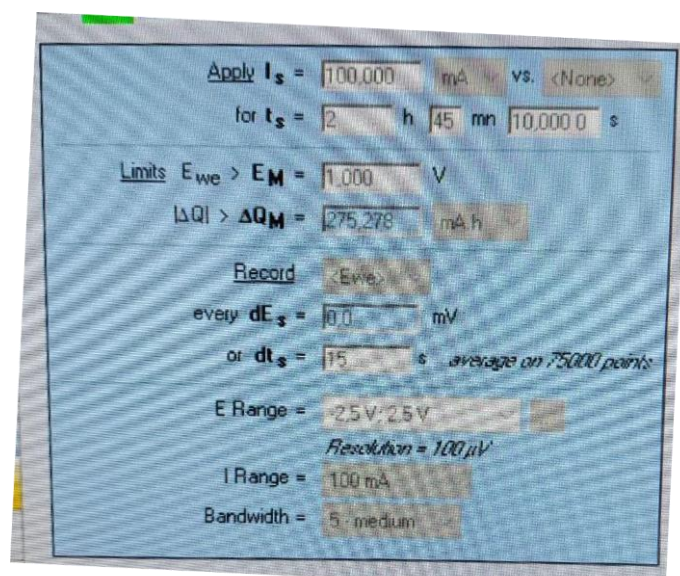


Figure 21 : Paramètres pour la première acquisition

PROJET TUTEURE MCPC

Nous pouvons alors tirer le tableau suivant :

N° de cycle	Temps charge + décharge	Tcharge exp/ Tcharge théo	N° de cycle	Temps charge + décharge	Tcharge exp/ Tcharge théo
0	18489,492	186,57%	28	9079,92	91,62%
1	9090,684	91,73%	29	9084,6	91,67%
2	9091,944	91,75%	30	9077,76	91,60%
3	9093,24	91,76%	31	9094,68	91,77%
4	9093,96	91,77%	32	9099,36	91,82%
5	9087,84	91,70%	33	9097,2	91,80%
6	9077,76	91,60%	34	9105,12	91,88%
7	9083,52	91,66%	35	9108,36	91,91%
8	9076,32	91,59%	36	9104,76	91,87%
9	9073,8	91,56%	37	9101,88	91,85%
10	9073,8	91,56%	38	9095,4	91,78%
11	9076,68	91,59%	39	9100,8	91,83%
12	9079,2	91,62%	40	9111,6	91,94%
13	9082,44	91,65%	41	9108	91,91%
14	9078,84	91,61%	42	9111,6	91,94%
15	9076,68	91,59%	43	9115,2	91,98%
16	9081	91,63%	44	9115,2	91,98%
17	9088,56	91,71%	45	9118,8	92,02%
18	9085,68	91,68%	46	9126	92,09%
19	9087,84	91,70%	47	9122,4	92,05%
20	9084,96	91,67%	48	9187,2	92,71%
21	9089,28	91,72%	49	6523,2	65,82%
22	9101,52	91,84%	50	11667,6	117,74%
23	9105,48	91,88%	51	9079,2	91,62%
24	9108	91,91%	52	9115,2	91,98%
25	9113,04	91,96%	53	9122,4	92,05%
26	9090,36	91,73%	54	9122,4	92,05%
27	9084,24	91,67%	55	9133,2	92,16%

Figure 22 : Rapport de temps de charge expérimentale/charge théorique en fonction du temps

Comme nous pouvons le voir, nous avons un rapport d'environ 91,81% alors que nous sommes censés avoir 100%, aussi nous avons des écarts aux cycles 49 et 50 (indiqués en rouge), et nous n'avons pour l'instant pas d'explication à cet écart.

PROJET TUTEURE MCPC

Malgré la constance de nos rapports dans le temps, nous pouvons les représenter dans la carte de contrôle suivante :

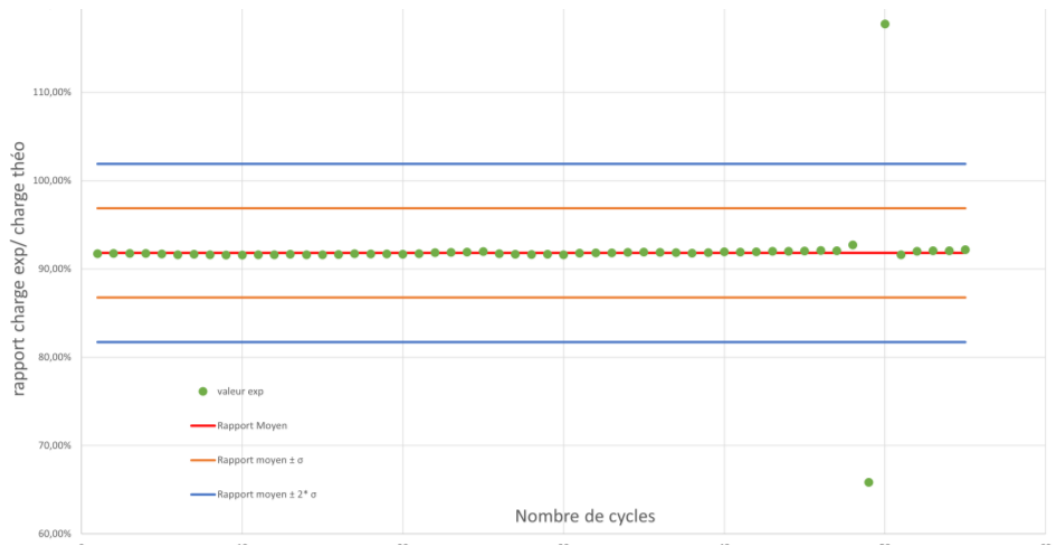


Figure 23 : Carte de contrôle charge expérimentale/charge théorique en fonction du temps

Nous observons bien les deux valeurs incohérentes. Afin de comprendre ces écarts nous pouvons étudier le rapport entre le temps de charge et celui de décharge de chaque cycle, tel que :

N° de cycle	Temps de charge en s	Temps de décharge en s	Rapport C/D	N° de cycle	Temps de charge en s	Temps de décharge en s	Rapport C/D
1	18489,492	27580,176	67,04%	15	272926,080	282010,680	96,78%
2	36672,120	45765,360	80,13%	16	291088,440	300183,120	96,97%
3	54859,320	63947,160	85,79%	17	309282,480	318379,680	97,14%
4	73024,920	82108,440	88,94%	18	327484,800	336593,160	97,29%
5	91184,760	100258,560	90,95%	19	345697,920	354799,800	97,43%
6	109332,360	118409,040	92,33%	20	363895,200	372996,000	97,56%
7	127488,240	136570,680	93,35%	21	382107,600	391215,600	97,67%
8	145649,520	154726,200	94,13%	22	400327,200	409442,400	97,77%
9	163807,200	172895,760	94,74%	23	418557,600	427676,400	97,87%
10	181981,440	191069,280	95,24%	24	436802,400	445924,800	97,95%
11	200154,240	209243,520	95,66%	25	455112,000	461635,200	98,59%
12	218345,040	227450,520	96,00%	26	473302,800	482382,000	98,12%
13	236558,520	245671,560	96,29%	27	491497,200	500619,600	98,18%
14	254761,920	263846,160	96,56%	28	509742,000	518875,200	98,24%

Figure 24 : Rapport de temps de charge/ temps de décharge en fonction du temps

Nous remarquons une augmentation du rapport entre le temps de charge et celui de décharge supposée être identiques.

Afin de l'observer plus clairement nous avons tracé la carte de contrôle suivante :

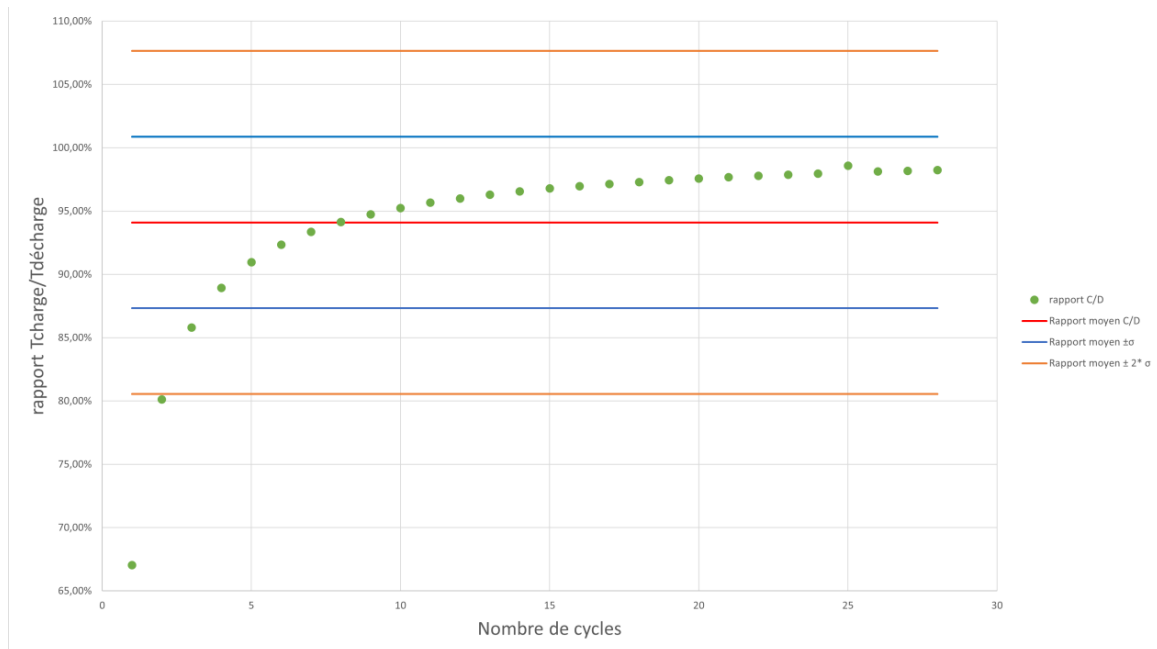


Figure 25 : Carte de contrôle de temps de charge/ temps de décharge en fonction du temps

Nous remarquons bien cette augmentation de ce rapport dans le temps avec une limite asymptotique à 100% qui est la valeur théorique que l'on est censée retrouver. La manipulation nous montre que l'expérience commence à être reproductible pour des temps supérieurs ou égales à 3 jours ce qui est intéressant dans le cas de stockage stationnaire. Cependant, on remarque que la stabilisation n'est pas immédiate ce qui est sûrement causé par le parasitage du dioxygène produit et de l'air présent dans les tuyaux reliant la pompe aux électrolytes.

En effet, ce rapport qui devient homogène nous montre une stabilisation au cours du temps.

Le fait que le système soit une batterie modèle (même couple redox à chaque pôle) nous aide à ne pas rencontrer de réactions parasites avec des molécules organiques et d'obtenir un modèle fiable au cours du temps.

4) Etalonnage de notre capteur de température

Pour ce qui est de la métrologie concernant la température, nous avons lié à notre spectromètre une PT100 de type B nous permettant de déterminer la température du milieu dans lequel on effectue les mesures, Aussi il nous a fallu d'abord nous assurer que notre PT100 affichait la bonne température. Puis nous avons déterminé son incertitude, nous permettant par la suite de pouvoir déterminer une incertitude globale à toute notre installation.

PROJET TUTEURE MCPC

Pour l'étalonnage de cette PT100, nous avons utilisé deux bains-marie étalonnés, le premier est un bain-marie à huile et le deuxième est un bain-marie à air. Le bain-marie à huile ayant une inertie importante, nous avons pu l'utiliser qu'à des températures supérieures à la température ambiante ($>25^{\circ}\text{C}$).

Aussi, c'est pour cela que pour l'incertitude finale, nous n'avons utilisé que les données déterminées à l'aide du bain-marie à air, ainsi les incertitudes correspondent aux écarts avec le bain-marie à air ainsi que la répétabilité de notre PT100.

Nous pouvons représenter les étalonnages avec chacun des deux bains-marie, tels que :

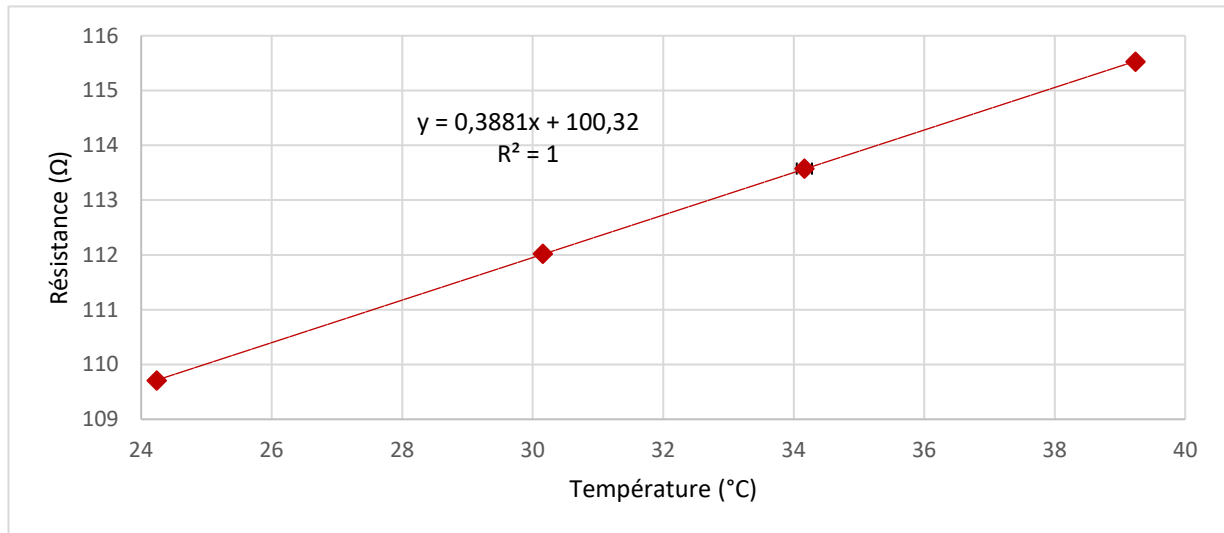


Figure 26 : Courbe d'étalonnage de la sonde Pt100 avec un bain-marie à huile

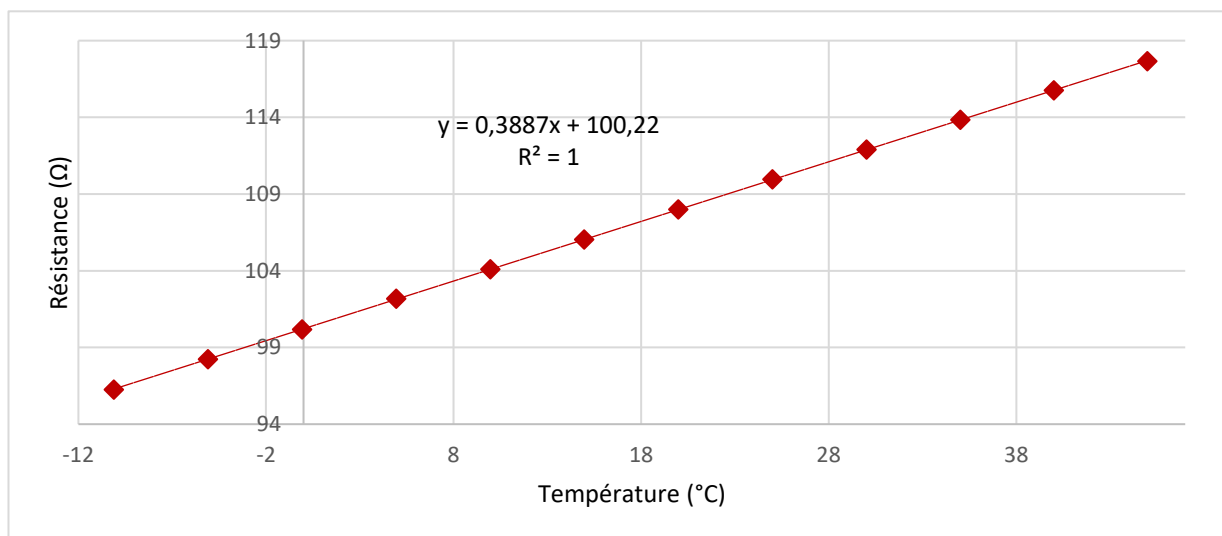


Figure 27 : Courbe d'étalonnage de la sonde Pt100 avec un bain-marie à air

Après calcul, on détermine bien une incertitude mais celle-ci est au maximum de $\pm 0,013^{\circ}\text{C}$, ce qui est faible. Ainsi, si le calcul de ϵ n'est pas bon par rapport au ϵ théorique, on saura que l'erreur est due au spectromètre.

III. Capteurs et logiciels

1°) Détermination des outils nécessaires

Pour rappel ce projet a pour but de suivre le fonctionnement de la batterie détaillé précédemment en fonction de temps et également pour avoir l'état de charge de cette dernière.

Pour cela nous avons décidé de programmer une interface en langage python car :

- Syntaxe claire et simple, ce qui le rend facile à apprendre et à utiliser pour les débutants en programmation
- Utilisé pour de nombreux types d'applications, notamment la science des données, l'apprentissage automatique, l'analyse de données, la programmation web, etc.
- Possède de nombreuses bibliothèques et frameworks puissants pour diverses tâches, y compris la visualisation de données, le traitement de données, etc.

Mais ce langage ne sera pas le seul, en effet nous avons besoin de programmer une partie du code en C++ qui sera compilé dans une carte Arduino Uno de chez GoTronic basée sur un ATmega328P cadencé à 16 MHz (Figure 23) car :

- Le C++ est un langage compilé qui peut être utilisé pour obtenir des performances plus élevées que les langages interprétés tels que Python. En raison de la nature temps réel de la mesure, les performances peuvent être cruciales pour garantir la précision des résultats.
- Les cartes Arduino ont généralement peu de mémoire disponible, il est donc préférable d'utiliser un langage qui permet un contrôle plus fin de l'utilisation de la mémoire. Le C++ permet de gérer manuellement la mémoire, ce qui permet d'optimiser l'utilisation de celle-ci et donc de réduire le coût du dispositif au lieu d'acheter une carte avec plus de mémoire mais donc plus chère.

Les logiciels utilisés pour effectués toute la programmation sont « Visual Studio Code » et « IDE Arduino »



Figure 28 : Arduino Uno Gotronic

Une fois que nous avons décidé de notre carte de développement il nous faut définir les grandeurs physiques nous permettant de calculer notre état de charge de la batterie (SOC).

Le suivi du fonctionnement de la batterie pouvait se faire en mesurant :

- Le pH de la solution
- La conductimétrie de la solution
- La température de la solution
- L'absorbance de la solution

Nous avons choisi de faire le suivi en fonction de ces deux derniers. Tout d'abord parce-que la température est la grandeur physique la plus simple à mesurer mais aussi parce qu'elle a un réel impact sur l'état de charge de la batterie. En effet, puisque comme démontrer précédemment, l'état de charge dépend de la concentration molaire.

$$A = \epsilon l c \Leftrightarrow c = \frac{A}{\epsilon l} \quad (19)$$

Or, la concentration dépend également de ϵ qui dépend elle-même de la température.

La température a donc un impact sur l'état de charge de la batterie et est donc une grandeur intéressante à choisir pour le suivi de la batterie.

Pour cela nous devons choisir des capteurs adaptés pour qu'ils s'intègrent parfaitement dans le dispositif et correspondent aux attentes de l'expérience.

Nous avons d'abord commencé à choisir la sonde de température car plus facile à programmer elle pourrait nous donner les premiers résultats relatifs à la batterie.

Cette sonde de température doit répondre à plusieurs critères ci-dessous :

- Prise de température dans une gamme de -15°C à 50°C
- Résistant au milieu liquide (étanche)
- Câble suffisamment long pour le plonger dans un électrolyte

Notre choix c'est donc orienter vers une sonde de température Pt100 IKE520/1M qui a les caractéristiques suivantes :

- Dimension du plongeur : $\varnothing 5,0 \times 20 \text{ mm}$
- Longueur : 1 m
- Classe B
- Précision à 0°C : $\pm 0,30^{\circ}\text{C}$
- Limites de température : -40 à 105°C
- Câble d'extension : $2 \times 0,30 \text{ mm}^2$
- Raccordement par fils double isolation

Cette sonde à toutes les caractéristiques requêt pour être intégrer dans notre système.

En effet, son petit diamètre permet de la faire passer dans les électrolytes, sa longueur acceptable permet de bien l'insérer dans cette dernière.

PROJET TUTEURE MCPC

La norme internationale CEI 751, dérivée de la norme DIN 43 760 qui définit les valeurs nominales ainsi que les écarts admissibles.

Les tables ont été établies notamment à partir d'une résistance de 100Ω à 0°C , d'où le terme Pt100 pour une sonde à résistance de platine dont la résistance est de 100Ω à 0°C . À 100°C la résistance est de $138,51\Omega$. La norme définit deux classes :

- Tolérance Classe B : $\pm (0,30 + 0,005 |t|)$ de -200°C à $+850^{\circ}\text{C}$
- Tolérance Classe A : $\pm (0,15 + 0,002 |t|)$ de -200°C à $+600^{\circ}\text{C}$

Avec : $|t|$ = valeur absolue de la température en $^{\circ}\text{C}$

La classe A donne une précision environ deux fois meilleure que la classe B. En règle générale, la classe B est d'usage industriel et la classe A est destinée aux laboratoires. Des tolérances plus resserrées sont parfois utilisées comme par exemple 1/3 Classe B dit encore 1/3 de DIN.

T en $^{\circ}\text{C}$	Rt en Ω	Coef. Temp. $\Omega/^{\circ}\text{C}$	Tolérance classe B		Tolérance classe A	
			T $^{\circ}\text{C}$	Rt Ω	T $^{\circ}\text{C}$	Rt Ω
-200	18,52	0,44	$\pm 1,3$	$\pm 0,56$	$\pm 0,55$	$\pm 0,24$
-100	60,26	0,41	$\pm 0,8$	$\pm 0,32$	$\pm 0,35$	$\pm 0,14$
0	100,00	0,39	$\pm 0,3$	$\pm 0,12$	$\pm 0,15$	$\pm 0,06$
100	138,51	0,38	$\pm 0,8$	$\pm 0,30$	$\pm 0,35$	$\pm 0,13$
200	175,86	0,37	$\pm 1,3$	$\pm 0,48$	$\pm 0,55$	$\pm 0,20$
300	212,05	0,35	$\pm 1,8$	$\pm 0,64$	$\pm 0,75$	$\pm 0,27$
400	247,09	0,34	$\pm 2,3$	$\pm 0,79$	$\pm 0,95$	$\pm 0,33$
500	280,98	0,33	$\pm 2,8$	$\pm 0,93$	$\pm 1,15$	$\pm 0,38$

Figure 29 : Extrait de la table de correspondance selon la norme CEI 751, amendement 2 de 1995, (d'après E. I. T. 90)

Mais dans notre cas la classe B est suffisamment précise, comme on peut le voir cette sonde pt100 à une précision de $\pm 0,30^{\circ}\text{C}$ à 0°C donc cela nous permet d'avoir une bonne valeur de la température et donc de l'état de charge de la batterie. De plus elle à une plage de prise de température allant de -40°C à 105°C ce qui est largement suffisant pour une utilisation extérieure.

La sonde Pt100 ne relève pas une température directement, elle relève une tension. C'est pour cela qu'il faut coupler la sonde avec le Adafruit MAX31865 qui est un module de capteur de température de résistance à plat (RTD) pour qui est utilisé avec des sondes de température PT100 ou PT1000.

Il utilise le circuit intégré MAX31865 de Maxim Integrated pour convertir la résistance de la sonde en une mesure de température et permet de connecter une sonde de température PT100 ou PT1000 à un microcontrôleur ou à un ordinateur pour mesurer la température dans une variété d'applications. Il est généralement utilisé pour la mesure de température de précision dans les systèmes de contrôle de processus, les équipements de laboratoire, les appareils ménagers et les applications industrielles.

PROJET TUTEURE MCPC

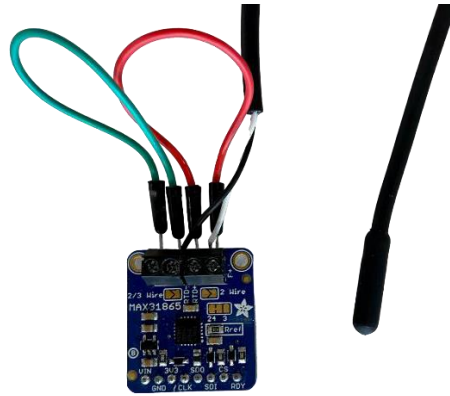
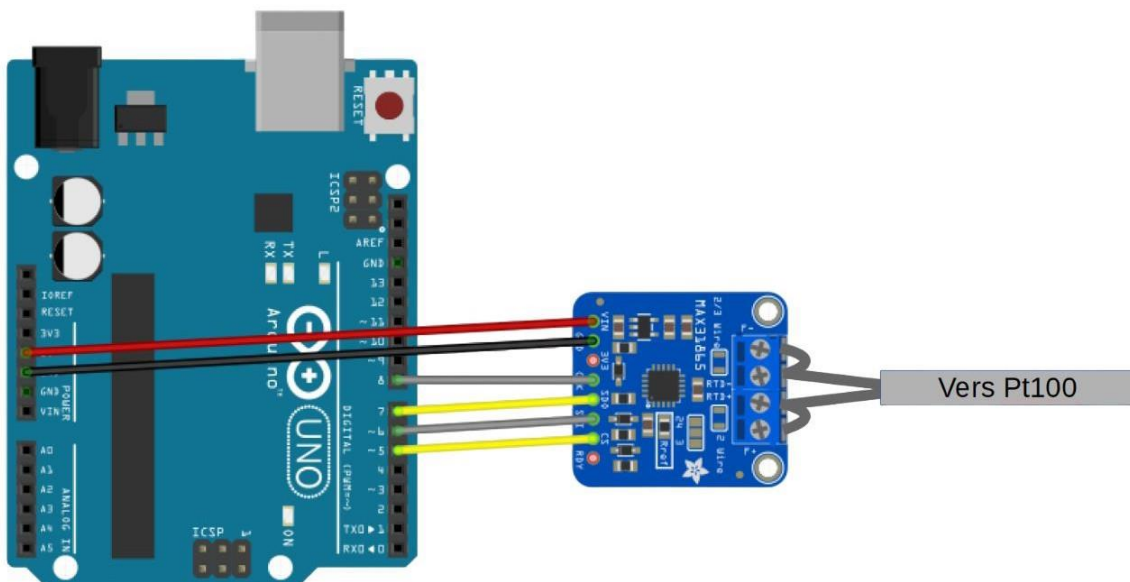


Figure 30 : Adafruit MAX31865 avec sonde Pt100

Nous avons la première partie de notre système pour la température mais pour que l'on reçoit des informations sur l'ordinateur il faut relier tous les composants ensemble de cette manière et ensuite connecter ce montage à l'ordinateur à l'aide d'un câble USB 2.0 A vers B.



Microcontrôleur	5V	GND	5	6	7	8
MAX31865	Vin	GND	CS	SDI	SDO	CLK

Figure 31 : Montage du microcontrôleur avec le système de température

Une fois le montage effectué nous devons étalonner la sonde Pt100 afin de garantir la précision, la conformité réglementaire, la traçabilité et la maintenance préventive des mesures de température.

En parallèle de cet étalonnage nous programmons une interface graphique avec des outils permettant par la suite d'avoir l'état de charge de la batterie en direct (voir partie III.2)

PROJET TUTEURE MCPC

Pour ce qui est de l'absorbance nous avons choisi un spectromètre miniature nommé Flame-S-UV-VIS-ES de la marque Ocean Insight (Figure 21) qui a les caractéristiques suivantes :

- Gamme de longueur d'onde : 200nm à 850nm
- Résolution : 1,34nm
- Temps d'intégration : 3,8 millisecondes à 10 secondes
- Dimensions : 89,1 mm x 63,3 mm x 31,9 mm (34,4 mm avec pieds)
- Masse : 265 grammes



Figure 32 : Spectromètre Flame-S-UV-VIS-ES de Ocean Insight

Plus d'informations sur :

<https://www.oceaninsight.com/products/spectrometers/general-purpose-spectrometer/flame-series/flame-t-uv-vis-es/?qty=1>

Pour le bon fonctionnement de ce spectromètre il est couplé avec un DH-mini (Figure 22) qui est une source de lumière UV-Vis-NIR (ultraviolet visible à proche infrarouge) à large bande avec une sortie intense de 200 à 2500 nm dans un boîtier compact. Il est doté de fonctionnalités telles que l'obturateur et d'une puissance de pointe il est le choix idéal pour notre choix par rapport à sa taille et à son coût



Figure 33 : Source de lumière DH-Mini de Ocean Optics (Vu de face, coté et arrière)

Plus d'informations sur :

<https://spectraservices.com/product/ocean-optics-dh-mini.html>

2°) Programmation

Pour visualiser les données nous devons constituer une interface graphique afin d'afficher et d'étudier ces dernières. Pour cela nous avons utilisé le logiciel QtDesigner qui permet de dessiner des fenêtres visuellement mais aussi de modifier les propriétés des widgets, d'utiliser des layouts, et d'effectuer la connexion entre signaux et slots ce qui simplifie grandement le travail car ce dernier génère automatiquement le code python (Il est quand même nécessaire de faire certaines modifications).

Nous avons nommé notre logiciel « TechVolt », lorsque nous le lançons il y a un écran de chargement le temps que l'ordinateur compile tous les dossiers nécessaires au bon fonctionnement du logiciel, nous avons défini ce temps d'attente à 1 seconde.

Ensuite l'écran de chargement se ferme pour laisser la place au logiciel principal. Nous avons un menu déroulant qui se trouve à gauche qui contient un écran de bord où par la suite nous retrouverons toutes les informations relatives à la batterie donc la température, l'absorbance et l'état de charge mais de façon simplifier.

Dans ce menu il y a également une fenêtre pour voir un graphique qui affiche la prise de la température en direct en fonction du temps avec la possibilité d'arrêter l'expérience ce qui lancera par la suite automatiquement une fenêtre pour sauvegarder les points de mesures prélevé dans un fichier Excel (csv) ou texte (txt) où le manipulateur pourra retraiter les informations à sa guise mais il y aura aussi un demi-cercle sous forme de barre de chargement qui changera de couleur au fur et à mesure que la température augmente pour mieux visualiser avec en son centre la température affichée littéralement.

Puis pour finir dans ce menu il y a la même fenêtre mais appliquer à l'absorbance. Nous n'avons pas encore d'idée claire pour permettre la visualisation simplifier et « complexe » car nous avons des difficultés à faire fonctionner le spectromètre, cela serait dû à des librairies trop vieilles et donc obsolètes.

Nous trouvons en haut à droite un menu avec une cloche qui sera par la suite le centre de notifications pour nous informer d'un problème ou le début et la fin d'une expérience programmer. On y trouve aussi trois icônes qui permettent de réduire, agrandir ou fermer le logiciel.

Puis en bas à droite nous trouvons un menu nommé « Paramètres » qui permettra par la suite (si nous avons le temps) de changer la langue ou le style noir et jaune en d'autre couleur plus claire comme le blanc.

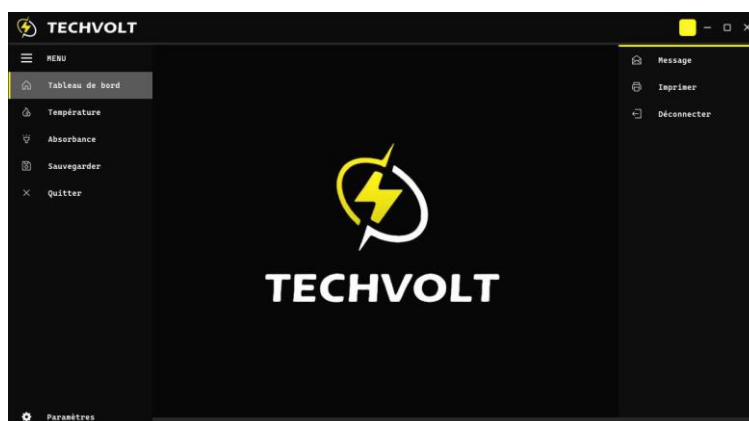


Figure 34 : Vue générale du logiciel TechVolt le 30/01/2023

PROJET TUTEURE MCPC

Pour programmer cette interface il a fallu :

- Installer et configuré l'environnement de développement (Visual Studio Code)
- Importer les librairies
- Ecrire les différents programmes à part
- Faire un programme principal pour simplifier
- Débogage au fur et à mesure que l'on intègre des outils

Une fois notre logiciel de codage installé correctement avec python nous pouvons commencer à programmer.

Pour cela nous importons des libraires Python (aussi appelé bibliothèques) qui sont des collections de modules qui fournissent des fonctionnalités supplémentaires pour les développeurs.

Elles peuvent être utilisées pour :

- ⇒ Réaliser des tâches courantes de manière plus rapide et plus efficace
- ⇒ Accéder à des bases de données, des services web et d'autres ressources en ligne
- ⇒ Implémenter des algorithmes complexes
- ⇒ Gérer des graphiques, des images et des données de visualisation
- ⇒ Intégrer des fonctionnalités telles que l'apprentissage automatique, la reconnaissance de la parole et le traitement du langage naturel.

```
1  import sys
2  import os
3  import platform
4
5  from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgi
6  from matplotlib.figure import Figure
7
8  from PySide6 import QtCore, QtGui, QtWidgets
9  from PySide6.QtCore import *
10 from PySide6.QtGui import *
11 from PySide6.QtWidgets import *
12
13 from splashscreen import *
14 from modules import *
15 from widgets import *
```

Figure 35 : Librairies utilisées dans le programme principal

Par exemple voici toutes les librairies utilisées dans le programme principale, on peut en citer quelques-unes comme :

Ligne 2 : La librairie « os » permet de créer des scripts pour automatiser des tâches système, de gérer des fichiers sur le système de fichiers et d'interagir avec le système d'exploitation de manière plus efficace.

Lignes 5 et 6 : La librairie « matplotlib » est utilisée pour la visualisation de données en science des données et dans d'autres domaines pour visualiser des tendances, identifier des relations et communiquer des résultats de manière claire et concise.

Lignes 8 à 11 : La librairie « PySide6 » est utilisé pour créer des applications de bureau pour Windows, macOS et Linux et peut être utilisé en conjonction avec d'autres bibliothèques pour créer des applications riches en fonctionnalités. (Nécessaire pour QtDesigner)

PROJET TUTEURE MCPC

Il est à noter que les 3 dernières librairies sont en réalité des raccourcis permettant d'accéder aux sous-programmes tels que pour la température que nous verrons juste après.

Maintenant que nous avons importé nos librairies nécessaires à notre projet nous pouvons utiliser QtDesigner pour créer notre fenêtre où l'on pourra imbriquer nos différents outils

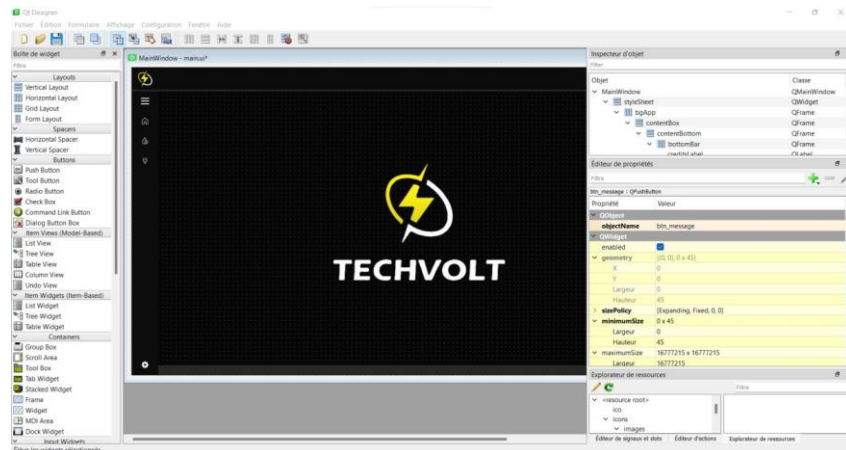


Figure 36 : Vue du logiciel TechVolt dans le logiciel QtDesigner

```
TechVolt V2.0 > modules > ui_main.py > Ui_MainWindow > setupUi
1 from PySide6.QtCore import *
2 from PySide6.QtGui import *
3 from PySide6.QtWidgets import *
4
5 from .resources_rc import *
6 from images.images import TechVolt_ico
7 from images.images import TechVolt_logo
8 from images.images import TechVolt_name
9
10 class Ui_MainWindow(object):
11     def setupUi(self, MainWindow):
12         if not MainWindow.setObjectName():
13             MainWindow.setObjectName(u"MainWindow")
14             MainWindow.resize(1280, 720)
15             MainWindow.setMinimumSize(QSize(940, 560))
16             self.setStyleSheet = QWidget(MainWindow)
17             self.setStyleSheet.setObjectName(u"styleSheet")
18             font = QFont()
19             font.setFamily(u"Cascadia Mono SemiBold")
20             font.setPointSize(10)
21             font.setBold(False)
22             font.setItalic(False)
23             font.setLegacyWeight(50)
24             self.setStyleSheet.setFont(font)
25             self.setStyleSheet.setStyleSheet(u"QWidget{\n"
26             "color: rgb(255, 255, 255);\n"
27             "font: 10pt \"Cascadia Mono SemiBold\";\n"
28             "}\n"
29             "\n"
30             "\n"
31             "/*Tooltip */\n"
32             "QToolTip {\n"
33             "color: #ffffff;\n"
34             "background-color: rgba(40, 40, 40, 180);\n"
35             "border: 1px solid rgb(44, 49, 58);\n"
36             "background-image: none;\n"
37             "background-position: left center;\n"
```

Figure 37 : Code généré par le logiciel QtDesigner

Ce code généré par QtDesigner est simplement l'aspect graphique avec les différents objets qui ont des classes qui permettront d'ajouter d'autres outils par la suite. A l'heure actuelle il fait 1000 lignes.

Maintenant que nous avons une fenêtre où nous pouvons afficher nos données nous pouvons ajouter d'autres fenêtres à l'intérieur ou d'autres objets pour nous permettre de voir la température, l'absorbance et l'état de charge. Malheureusement aujourd'hui nous n'avons pas régler ce problème

PROJET TUTEURE MCPC

car trop complexe pour notre niveau donc il est étudié par un enseignant référent du projet (Mr. Quiquempois).

Mais nous pouvons tout de même faire les programmes à part, c'est pour cela que nous avons créé une fenêtre matplotlib afin de voir la température en direct à partir des données reçues de notre montage avec la carte Arduino Uno, le MAX31865 et la Pt100.

On a d'abord fait le code Arduino donc en C++ et nous l'avons adapté au langage python afin que la carte Arduino et l'ordinateur puisse communiquer ensemble :

```
1 ////////////////////////////////////////////////// BIBLIOTHEQUES //////////////////////////////////////
2 #include <Adafruit_MAX31865.h>
3 Adafruit_MAX31865 mon_capteur = Adafruit_MAX31865(5, 6, 7, 8);
4
5
6
7 unsigned long timer = 0;
8 long loopTime = 5000;
9
10 ////////////////////////////////////////////////// INITIALISATION //////////////////////////////////////
11 void setup() {
12     mon_capteur.begin(MAX31865_2WIRE);
13     Serial.begin(115200);
14     timer = micros();
15 }
16 ////////////////////////////////////////////////// MESURE EFFECTUEE PAR LA SONDE //////////////////////////////////////
17 void loop() {
18     timeSync(loopTime);
19     float R = mon_capteur.lecture_resistance();
20     double val = (R-100.22)/0.388;
21     sendToPC(&val);
22 }
23
24 void timeSync(unsigned long deltaT)
25 {
26     unsigned long currTime = micros();
27     long timeToDelay = deltaT - (currTime - timer);
28     if (timeToDelay > 5000)
29     {
30         delay(timeToDelay / 1000);
31         delayMicroseconds(timeToDelay % 1000);
32     }
33     else if (timeToDelay > 0)
34     {
35         delayMicroseconds(timeToDelay);
36     }
37     else
38     {
39         // timeToDelay est négatif donc nous commençons immédiatement
40     }
41     timer = currTime + timeToDelay;
42 }
43 ////////////////////////////////////////////////// COMMUNICATION DES DONNEES //////////////////////////////////////
44 void sendToPC(int* data)
45 {
46     byte* byteData = (byte*)(data);
47     Serial.write(byteData, 2);
48 }
49
50 void sendToPC(double* data)
51 {
52     byte* byteData = (byte*)(data);
53     Serial.write(byteData, 4);
54 }
```

Figure 38 : Code Arduino pour sonde de température

Voici le code pour Arduino qui lit les valeurs de température à partir d'un thermistor et envoie les données par communication série à un PC.

Le thermistor est interfacé avec un amplificateur de thermocouple Adafruit MAX31865, qui est initialisé dans la fonction setup().

La fonction loop() effectue la mesure de la température toutes les 5 secondes en appelant la fonction timeSync(), qui synchronise l'intervalle de mesure.

La valeur de la température est calculée à partir de la résistance du thermistor et envoyée au PC en appelant la fonction sendToPC() avec un pointeur vers la valeur de température.

La fonction sendToPC() prend deux arguments, un pour les valeurs int et un autre pour les valeurs double, et envoie les données par communication série sous forme de données brutes.

PROJET TUTEURE MCPC

Voici le code qui permet de faire un graphique pour afficher la température en direct à partir des données renvoyer par une sonde Pt100 :

```

1 report_timeval
2
3 report_serializable as int
4 report_serializable as float
5 report_serializable as string
6
7 report_timeval
8 report_collections
9 report_point as pt
10
11 from threading import Thread
12
13 # Expression de la barre d'outil Antialias
14
15 #####
16 # 1 Le PROGRAMME ne fait QUE des TOUTES !!! Antialias ne fonctionne PAS !!!
17 #####
18
19 class Polyline:
20     def __init__(self, serialPort = "COM3", serialSpeed = 115200, plotLength = 100, dataNumBytes = 4):
21         self.port = serialPort
22         self.baud = serialSpeed
23         self.plotLength = plotLength
24         self.dataNumBytes = dataNumBytes
25         self.data = bytearray(dataNumBytes)
26         self.data = collection.deque([0] * plotLength, maxlen=plotLength)
27         self.isDrawing = False
28         self.thread = None
29         self.plotArea = 0
30         self.previousValue = 0
31         self.counter = 1
32
33 ##### VERIFICATION CONNECTION #####
34
35 print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
36 try:
37     serialConnection = serial.Serial(serialPort, serialSpeed, timeout=0.3)
38     print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
39 except:
40     print('Impossible de se connecter à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
41
42 ##### DEFINITION DES VARIABLES #####
43
44 def readSerializable(self):
45     self.thread = None
46     self.thread = Thread(target=self.backgroundThread)
47     self.thread.start()
48
49 while self.isDrawing is True:
50     time.sleep(0.3)
51
52 def getSerializable(self, fromLine, lineLabelList, lineLabel, timePlot):
53     serialLine = line.get_line()
54     self.plotArea = line.get_plotArea()
55     self.previousValue = self.previousValue + 1000
56     serialConnection.write(serialLine)
57     timePlot.set_label('valeur : ' + str(self.plotArea) + ' m')
58     while 1:
59         data = serialLine.read(self.dataNumBytes)
60         self.data.append(data)
61         line.set_data(line.get_plotLength(), self.data)
62         lineLabel.set_label('valeur : ' + str(round(value, 2))
63                             + self.counter * round(self.data, 1))
64
65 def backgroundThread(self):
66     time.sleep(0.3)
67     self.dataNumBytes = read_serial_buffer()
68     while self.isPlot:
69         self.serialConnection.read(self.dataNumBytes)
70         self.isDrawing = True
71
72 def close(self):
73     self.isPlot = False
74     self.thread.interrupt()
75     print('Connexion : ')
76
77 #####
78
79 print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
80 try:
81     serialConnection = serial.Serial(serialPort, serialSpeed, timeout=0.3)
82     print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
83 except:
84     print('Impossible de se connecter à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
85
86 ##### DEFINITION DES VARIABLES #####
87
88 def readSerializable(self):
89     self.thread = None
90     self.thread = Thread(target=self.backgroundThread)
91     self.thread.start()
92
93 while self.isDrawing is True:
94     time.sleep(0.3)
95
96 def getSerializable(self, fromLine, lineLabelList, lineLabel, timePlot):
97     serialLine = line.get_line()
98     self.plotArea = line.get_plotArea()
99     self.previousValue = self.previousValue + 1000
100     serialConnection.write(serialLine)
101     timePlot.set_label('valeur : ' + str(self.plotArea) + ' m')
102     while 1:
103         data = serialLine.read(self.dataNumBytes)
104         self.data.append(data)
105         line.set_data(line.get_plotLength(), self.data)
106         lineLabel.set_label('valeur : ' + str(round(value, 2))
107                             + self.counter * round(self.data, 1))
108
109 def backgroundThread(self):
110     time.sleep(0.3)
111     self.dataNumBytes = read_serial_buffer()
112     while self.isPlot:
113         self.serialConnection.read(self.dataNumBytes)
114         self.isDrawing = True
115
116 def close(self):
117     self.isPlot = False
118     self.thread.interrupt()
119     print('Connexion : ')
120
121 #####
122
123 print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
124 try:
125     serialConnection = serial.Serial(serialPort, serialSpeed, timeout=0.3)
126     print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
127 except:
128     print('Impossible de se connecter à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
129
130 ##### DEFINITION DES VARIABLES #####
131
132 def readSerializable(self):
133     self.thread = None
134     self.thread = Thread(target=self.backgroundThread)
135     self.thread.start()
136
137 while self.isDrawing is True:
138     time.sleep(0.3)
139
140 def getSerializable(self, fromLine, lineLabelList, lineLabel, timePlot):
141     serialLine = line.get_line()
142     self.plotArea = line.get_plotArea()
143     self.previousValue = self.previousValue + 1000
144     serialConnection.write(serialLine)
145     timePlot.set_label('valeur : ' + str(self.plotArea) + ' m')
146     while 1:
147         data = serialLine.read(self.dataNumBytes)
148         self.data.append(data)
149         line.set_data(line.get_plotLength(), self.data)
150         lineLabel.set_label('valeur : ' + str(round(value, 2))
151                             + self.counter * round(self.data, 1))
152
153 def backgroundThread(self):
154     time.sleep(0.3)
155     self.dataNumBytes = read_serial_buffer()
156     while self.isPlot:
157         self.serialConnection.read(self.dataNumBytes)
158         self.isDrawing = True
159
160 def close(self):
161     self.isPlot = False
162     self.thread.interrupt()
163     print('Connexion : ')
164
165 #####
166
167 print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
168 try:
169     serialConnection = serial.Serial(serialPort, serialSpeed, timeout=0.3)
170     print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
171 except:
172     print('Impossible de se connecter à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
173
174 ##### DEFINITION DES VARIABLES #####
175
176 def readSerializable(self):
177     self.thread = None
178     self.thread = Thread(target=self.backgroundThread)
179     self.thread.start()
180
181 while self.isDrawing is True:
182     time.sleep(0.3)
183
184 def getSerializable(self, fromLine, lineLabelList, lineLabel, timePlot):
185     serialLine = line.get_line()
186     self.plotArea = line.get_plotArea()
187     self.previousValue = self.previousValue + 1000
188     serialConnection.write(serialLine)
189     timePlot.set_label('valeur : ' + str(self.plotArea) + ' m')
190     while 1:
191         data = serialLine.read(self.dataNumBytes)
192         self.data.append(data)
193         line.set_data(line.get_plotLength(), self.data)
194         lineLabel.set_label('valeur : ' + str(round(value, 2))
195                             + self.counter * round(self.data, 1))
196
197 def backgroundThread(self):
198     time.sleep(0.3)
199     self.dataNumBytes = read_serial_buffer()
200     while self.isPlot:
201         self.serialConnection.read(self.dataNumBytes)
202         self.isDrawing = True
203
204 def close(self):
205     self.isPlot = False
206     self.thread.interrupt()
207     print('Connexion : ')
208
209 #####
210
211 print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
212 try:
213     serialConnection = serial.Serial(serialPort, serialSpeed, timeout=0.3)
214     print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
215 except:
216     print('Impossible de se connecter à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
217
218 ##### DEFINITION DES VARIABLES #####
219
220 def readSerializable(self):
221     self.thread = None
222     self.thread = Thread(target=self.backgroundThread)
223     self.thread.start()
224
225 while self.isDrawing is True:
226     time.sleep(0.3)
227
228 def getSerializable(self, fromLine, lineLabelList, lineLabel, timePlot):
229     serialLine = line.get_line()
230     self.plotArea = line.get_plotArea()
231     self.previousValue = self.previousValue + 1000
232     serialConnection.write(serialLine)
233     timePlot.set_label('valeur : ' + str(self.plotArea) + ' m')
234     while 1:
235         data = serialLine.read(self.dataNumBytes)
236         self.data.append(data)
237         line.set_data(line.get_plotLength(), self.data)
238         lineLabel.set_label('valeur : ' + str(round(value, 2))
239                             + self.counter * round(self.data, 1))
240
241 def backgroundThread(self):
242     time.sleep(0.3)
243     self.dataNumBytes = read_serial_buffer()
244     while self.isPlot:
245         self.serialConnection.read(self.dataNumBytes)
246         self.isDrawing = True
247
248 def close(self):
249     self.isPlot = False
250     self.thread.interrupt()
251     print('Connexion : ')
252
253 #####
254
255 print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
256 try:
257     serialConnection = serial.Serial(serialPort, serialSpeed, timeout=0.3)
258     print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
259 except:
260     print('Impossible de se connecter à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
261
262 ##### DEFINITION DES VARIABLES #####
263
264 def readSerializable(self):
265     self.thread = None
266     self.thread = Thread(target=self.backgroundThread)
267     self.thread.start()
268
269 while self.isDrawing is True:
270     time.sleep(0.3)
271
272 def getSerializable(self, fromLine, lineLabelList, lineLabel, timePlot):
273     serialLine = line.get_line()
274     self.plotArea = line.get_plotArea()
275     self.previousValue = self.previousValue + 1000
276     serialConnection.write(serialLine)
277     timePlot.set_label('valeur : ' + str(self.plotArea) + ' m')
278     while 1:
279         data = serialLine.read(self.dataNumBytes)
280         self.data.append(data)
281         line.set_data(line.get_plotLength(), self.data)
282         lineLabel.set_label('valeur : ' + str(round(value, 2))
283                             + self.counter * round(self.data, 1))
284
285 def backgroundThread(self):
286     time.sleep(0.3)
287     self.dataNumBytes = read_serial_buffer()
288     while self.isPlot:
289         self.serialConnection.read(self.dataNumBytes)
290         self.isDrawing = True
291
292 def close(self):
293     self.isPlot = False
294     self.thread.interrupt()
295     print('Connexion : ')
296
297 #####
298
299 print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
300 try:
301     serialConnection = serial.Serial(serialPort, serialSpeed, timeout=0.3)
302     print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
303 except:
304     print('Impossible de se connecter à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
305
306 ##### DEFINITION DES VARIABLES #####
307
308 def readSerializable(self):
309     self.thread = None
310     self.thread = Thread(target=self.backgroundThread)
311     self.thread.start()
312
313 while self.isDrawing is True:
314     time.sleep(0.3)
315
316 def getSerializable(self, fromLine, lineLabelList, lineLabel, timePlot):
317     serialLine = line.get_line()
318     self.plotArea = line.get_plotArea()
319     self.previousValue = self.previousValue + 1000
320     serialConnection.write(serialLine)
321     timePlot.set_label('valeur : ' + str(self.plotArea) + ' m')
322     while 1:
323         data = serialLine.read(self.dataNumBytes)
324         self.data.append(data)
325         line.set_data(line.get_plotLength(), self.data)
326         lineLabel.set_label('valeur : ' + str(round(value, 2))
327                             + self.counter * round(self.data, 1))
328
329 def backgroundThread(self):
330     time.sleep(0.3)
331     self.dataNumBytes = read_serial_buffer()
332     while self.isPlot:
333         self.serialConnection.read(self.dataNumBytes)
334         self.isDrawing = True
335
336 def close(self):
337     self.isPlot = False
338     self.thread.interrupt()
339     print('Connexion : ')
340
341 #####
342
343 print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
344 try:
345     serialConnection = serial.Serial(serialPort, serialSpeed, timeout=0.3)
346     print('Connexion à : ' + str(serialPort) + ' à ' + str(serialSpeed) + ' Baud')
347 except:
348    
```

Figure 39 : Code Python pour graphique en direct de la température en fonction du temps

Le code ci-dessus est un outil de visualisation de données pour les lectures de température reçues à partir d'une connexion série. Il utilise les bibliothèques matplotlib et série de Python pour recevoir des données du port série spécifié dans la variable "portName" et affiche les lectures de température dans un graphique en temps réel. Le graphique se met à jour à chaque fois que de nouvelles données sont reçues, avec une certaine valeur de latence. Les données sont également enregistrées dans un fichier CSV (Excel) lorsque le script est fermé. Il faut noter que le script ne fonctionne que sur Windows, comme indiqué dans le commentaire en haut à la ligne 15.

Voici le graphique de la visualisation de la température :

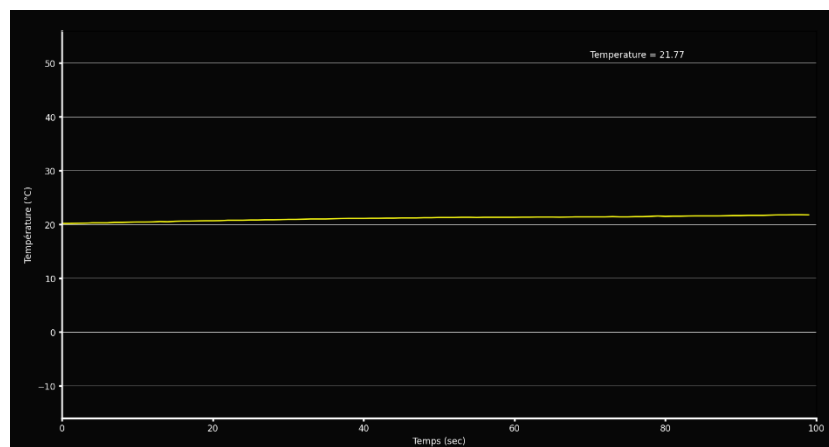


Figure 40 : Visualisation de la température

Après cette étape il nous faut faire le programme pour le spectromètre et tout réunir dans la même interface pour visualiser les données et les sauvegarder mais comme dit précédemment, nous sommes heurtées à certaines difficultés, voici la liste des problèmes que nous devons régler :

- ⇒ Avoir la fenêtre Matplotlib dans l'onglet température pour afficher cette dernière en direct (actualiser le graphique) + bouton tracer qui fonctionne
- ⇒ Avoir un bouton "Sauvegarder" qui permet de sauvegarder les données du graphique sur fichier txt et/ou csv avec maximum 2 décimales à l'intérieur
- ⇒ Librairie pyUSB qui ne fonctionne pas pour le spectromètre

Il faut prendre en compte que certains de ces problèmes sont déjà partiellement résolus comme la sauvegarde des données avec le nom de fichier que l'on souhaite et n'avoir que deux décimales car suffisant vu la précision du capteur.

Le programme du spectromètre utilise la bibliothèque Seabreeze pour communiquer avec un spectromètre. Il utilise également la bibliothèque Matplotlib pour créer un graphique à partir des données mesurées par le spectromètre. Le premier bloc de code importe les bibliothèques nécessaires pour le programme. Seabreeze est utilisé pour communiquer avec le spectromètre, Spectrometer est une classe qui permet d'interagir avec le spectromètre, et Matplotlib est utilisé pour créer le graphique.

La ligne suivante crée une instance de Spectrometer à partir du premier spectromètre disponible. Cette ligne peut être modifiée pour se connecter à un spectromètre spécifique en utilisant son nom ou son identifiant. Ensuite il imprime des informations sur le spectromètre, y compris son nom, son identifiant et la longueur d'onde de référence et il définit le temps d'intégration à 1 seconde (1 000 000 microsecondes).

Après cela il récupère les longueurs d'onde et les intensités mesurées pour finir par tracer un graphique des intensités en fonction des longueurs d'onde avec la fonction plot() de Matplotlib. Puis enfin, la ligne plt.show() affiche le graphique.

```
1  ##### BIBLIOTHEQUE #####
2  import seabreeze
3  from seabreeze.spectrometers import Spectrometer
4  import matplotlib.pyplot as plt
5
6  ##### CREATION INSTANCE #####
7  spec = Spectrometer.from_first_available()
8  print(spec)
9
10 ##### TEMPS D'INTEGRATION #####
11 spec.integration_time_micros(1000000)
12
13 ##### RECUPERATION DONNEES #####
14 wavelengths = spec.wavelengths()
15 intensities = spec.intensities()
16
17 ##### CREATION GRAPHIQUE #####
18 plt.plot(wavelengths,intensities)
19 plt.show()
```

Figure 41 : Code Python pour graphique du spectromètre (spectre)

Nous pouvons voir sur la Figure 36 un spectre qui a été effectué vers la couleur bleue et la Figure 37 effectué vers la couleur rouge. On remarque les caractéristiques des couleurs c'est-à-dire leur longueur d'onde respective avec leurs intensités :

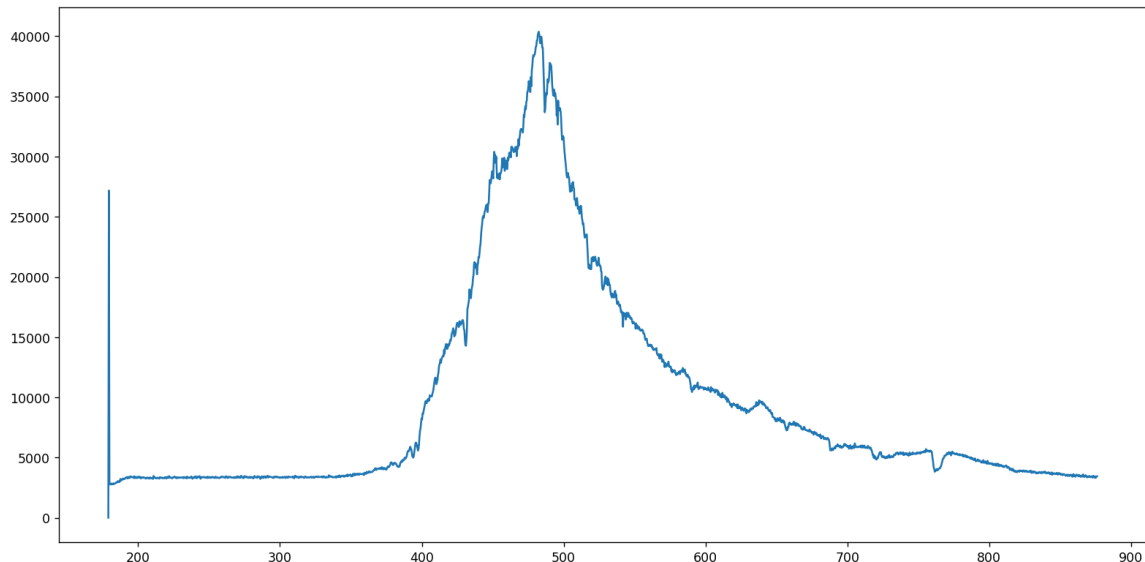


Figure 42 : Spectre effectué vers de la couleur bleue

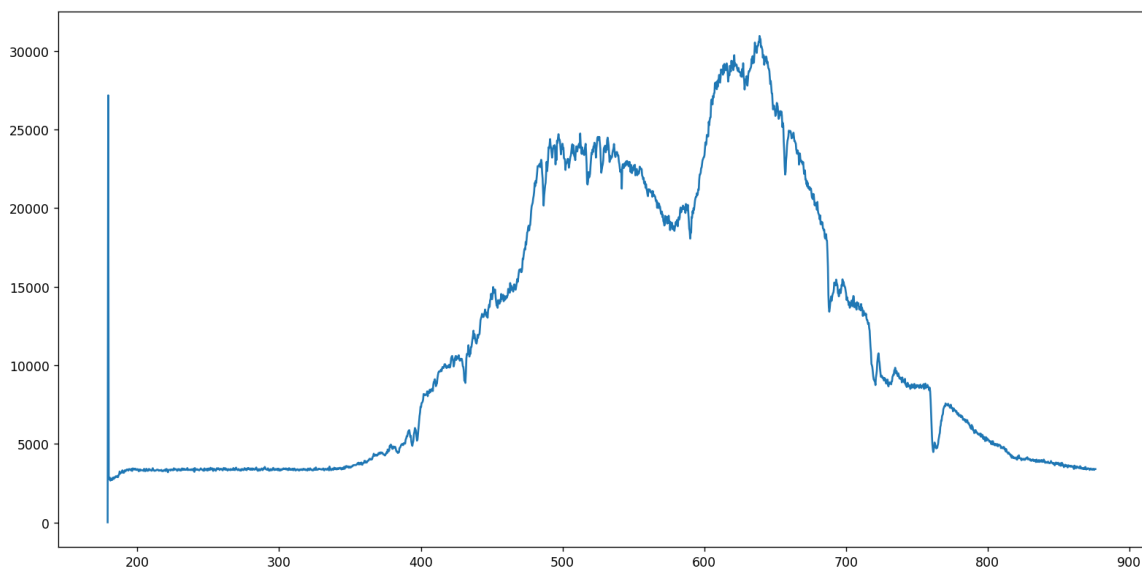


Figure 43 : Spectre effectué vers de la couleur rouge

Malheureusement par difficultés de programmation nous n'avons pas réussi à intégrer ces graphiques dans l'interface et à en extraire les données pour ensuite les réinjecter dans un algorithme permettant de calculer l'état de charge.

IV. Conclusion

Dans ce projet, notre objectif était de suivre le fonctionnement d'une batterie de type *red ox* au cours du temps et en fonction d'un paramètre comme la température, notamment en étudiant son état de charge.

Ce projet nous a permis de mieux comprendre les batteries Redox en général, de calculer efficacement l'état de charge d'une batterie et le temps de charge théorique.

Il nous a permis de mieux comprendre la spectrométrie en générale, que ça soit expérimentalement ou théoriquement.

De plus nous avons même pu faire de la programmation pour créer une interface graphique permettant de pouvoir piloter à distance notre batterie ainsi que de pouvoir récolter et traiter les données d'état de charge de la batterie.

Aussi nous avons étudié le comportement de certaines grandeurs comme l'absorbance en fonction de la température.

Cependant nous n'avons pas réussi à allier la température et l'absorbance ensemble dans notre installation et nous n'avons pas pu afficher les résultats que nous voulions (ex : état de charge) sur notre interface graphique dû à des problèmes de programmation.

Aussi, par manque de données, nous n'avons pas pu définir d'incertitude finale concernant l'état de charge.

Le but du projet est donc partiellement atteint.

Aussi par la suite, nous pourrons effectuer ces mesures. Ce qui nous permettra de trouver ce qu'il nous manque à l'heure actuelle. Nous pourrons étudier la réaction de la batterie si nous venons à modifier d'autres grandeurs telles que la conductivité, l'humidité ou encore le débit.

Pour finir, d'un point de vue plus personnel, ce projet nous a permis de cultiver notre capacité à travailler en groupe. Il nous a appris à se répartir les tâches et malgré les difficultés que nous avons eu par rapport à cela, nous pouvons aujourd'hui comprendre ce que nous avons mal fait.

V. Bibliographie :

- Extrait de « Experimental Investigation of Air relative Humidity γ (RH) Cycling Tests on MEA/Cell Aging in PEMFC Part I: Study of High RH Cycling Test With air RH at 62%/100% » B. T. Huang , Y. Chatillon , C. Bonnet, F. Lapique , S. Leclerc , M. Hinaje , S. Raël
- APPROCHES EXPERIMENTALES ET ANALYSE PROBABILISTE POUR LE DIAGNOSTIC DE PILES A COMBUSTIBLE DE TYPE PEM. Sébastien Wasterlain
- Méthodes électrochimiques pour la caractérisation des piles à combustibles de type PEM en empilement
Yohann Chatillon
- « De la chimie des solutions à l'électrochimie » Thermodynamique et cinétique électrochimiques de Didier Devilliers, Emmanuel Briot, Denise Krulic et Eric Mahé

