TP METHODES NUMERIQUES ET IA - CORRIGE

FONCTIONNEMENT PROPOSE :

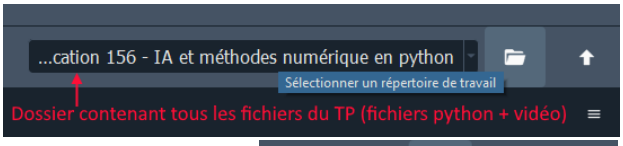
Ce TP est utilisé en TP en CPGE PT. Il s'étale sur **2 séances de TP de 2h20**. Les élèves travaillent chacun devant un poste informatique (**travail individuel**). Ils regroupent leurs conclusions dans un document réponse. L'enseignant passe vérifier les avancées de chaque élève et effectue des **mises en commun / corrections périodiques** via vidéo-projection des étapes du travail.

ACTIVITE 1 – ACQUISITION DES DONNEES PAR TRACKING

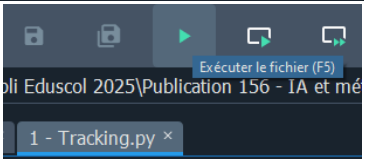
OUVERTURE ET EXECUTION DU FICHIER PYTHON "1 - TRACKING.PY"

La procédure est volontairement très détaillée afin de guider les plus néophytes en python.

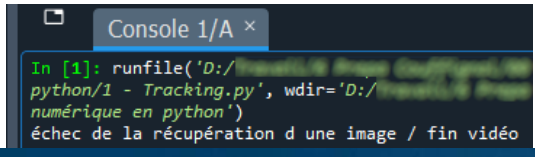
- Installer si ce n'est pas encore fait l'interface de développement Spyder IDE (téléchargeable gratuitement – on conseille d'installer WinPython qui inclut cette interface et le compilateur python associé).
- Lancer Spyder et sélectionner en haut à droite le répertoire de travail = dossier contenant les fichiers python et la vidéo à traiter.

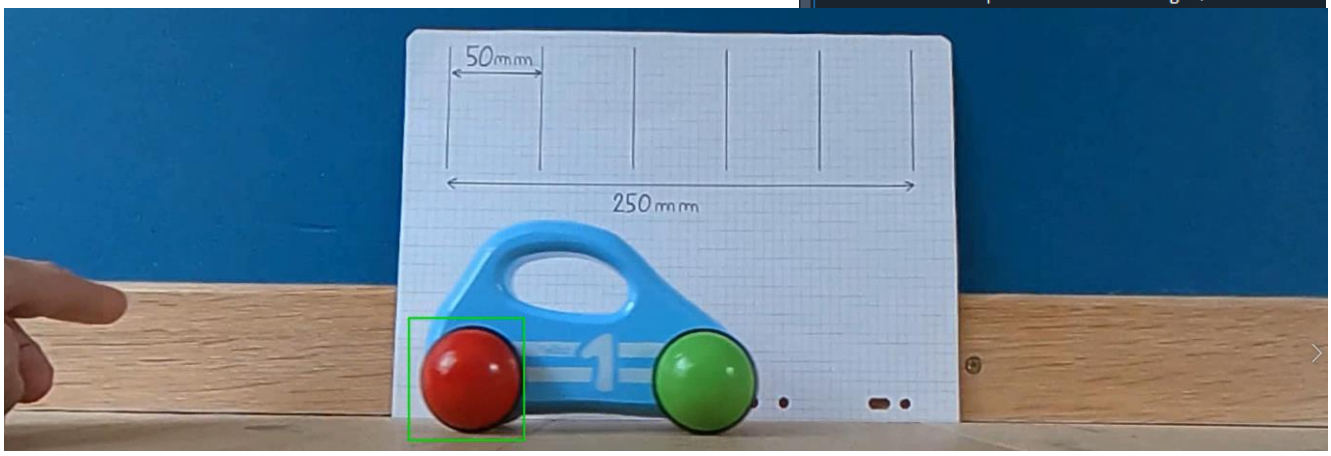


- Ouvrir (fichier – ouvrir) le fichier "1 – Tracking.py".
- Exécuter le fichier en cliquant sur la flèche verte.



- Une fenêtre s'ouvre, présentant la première image de la vidéo : cliquer pour définir un ROI (Region Of Interest = zone à suivre durant le tracking) de forme rectangulaire, autour d'une des roues ou de la voiture complète par exemple. Une fois le ROI défini, appuyer sur Entrée et observer le suivi du ROI effectué par l'algorithme. A la fin, la fenêtre vidéo se ferme automatiquement et le message " échec de la récupération d une image / fin vidéo" apparaît dans la console (l'algorithme ne trouve plus d'image suivante lorsqu'il arrive en fin de vidéo).





Remarque : le tracking fonctionne plus ou moins bien selon le ROI choisi.

- Le code crée automatiquement dans le répertoire de travail :
  - Une vidéo "tracked.avi" présentant le suivi du ROI superposé à la vidéo initiale.
  - Un fichier "image1.jpg" correspondant à la première image de la vidéo et servant pour la suite du TP (mise à l'échelle pixels → m).

- Vérifier dans l'explorateur de variables que les 3 variables suivantes ont bien été créées :

time	Array of float64	(469,)	[0.
tracker	TrackerKCF	1	Tracker
video	VideoCapture	1	VideoCa
w	int	1	172
x	int	1	1485
xtrack	Array of float64	(469,)	[97.
y	int	1	510
ytrack	Array of float64	(469,)	[492.

- *Time* : vecteur correspondant aux instants des prises de vue (déduit du framerate)
- *xtrack* : vecteur contenant la position horizontale du centre du ROI
- *ytrack* : vecteur contenant la position verticale du centre du ROI

Remarque : la dimension de ces vecteurs correspond au nombre d'images de la vidéo.

REMARQUE SUR LA VIDEO TRAITEE :

La vidéo proposée ici a été enregistrée avec une caméra type GoPro (résolution 1920 x 1080, 240 images / s) qui permet d'obtenir des résultats satisfaisants même avec un mouvement relativement rapide. Il est également possible d'enregistrer une vidéo avec un smartphone et d'effectuer une analyse similaire. On fera dans ce cas attention aux points suivants :

- **Conditions d'éclairage** favorables (fond uni dans l'idéal) : contrastes importants, pas (peu) d'ombres.
- **Caméra / smarphone fixe** par rapport au sol (ou par rapport au référentiel par rapport auquel on effectue le tracking) : une caméra tenue en main ne permettra pas de résultats satisfaisants
- Mouvement de **l'objet occupant la totalité de la zone de prise de vue** pour obtenir une résolution satisfaisante sur la mesure de position.
- **Réglages de prise de vue** : pour un mouvement relativement rapide, on privilégiera un framerate (nombre d'images par seconde) élevé, sans pour autant avoir de flou sur l'image (absence de flou favorisée par un bon éclairage). Pour un mouvement plus lent, on privilégiera une résolution de prise de vue élevée. Dans l'idéal, on souhaite une résolution d'image et un framerate élevés, ce qui justifie l'utilisation d'une caméra type GoPro, mais la plupart des smartphones actuels permettent également une acquisition à des framerates élevés (120 ou 240 images/s).
- **Déplacement plan et parallèle au plan focal** de l'objectif de la caméra.
- **Elément de mise à l'échelle :** insérer dans le plan de mouvement un objet dont la dimension est connue afin d'effectuer la mise à l'échelle pixels → m.

Q1 : *A quelle caractéristique du ROI correspondent $xtrack$ et $ytrack$?*

A la position du centre du ROI (rectangle).

Quel point sert de référence à cette mesure ?

Le point en bas à gauche de l'image sert de référence.

Quelles sont les unités de ces variables ?

Les positions sont relevées en pixels.

Q2 : *Quelles sont les unités de la variable $time$?*

Temps relevé en secondes

A partir de quelle information est-elle incrémentée ?

Incrémentation élaborée à partir du framerate (nbre d'images par secondes) récupéré dans les données EXIF de la vidéo (visibles dans les propriétés du fichier depuis un gestionnaire de fichiers).

Q3 : *Commenter les performances du tracking effectué :*

Le tracking est effectué correctement, même si on remarque sur certaines images un décalage du ROI récupéré par rapport à la position réelle de la partie à tracker. On note aussi des "oscillations" d'une image à l'autre.

Sources d'erreur de la mesure effectuée ?

On citera principalement les déformations induites par l'objectif de la caméra. De plus, la mesure servant de référence de distance (utilisée dans l'activité 2) n'est pas totalement dans le même plan que le centre de la voiture. Les reflets changeants perturbent également le tracking.

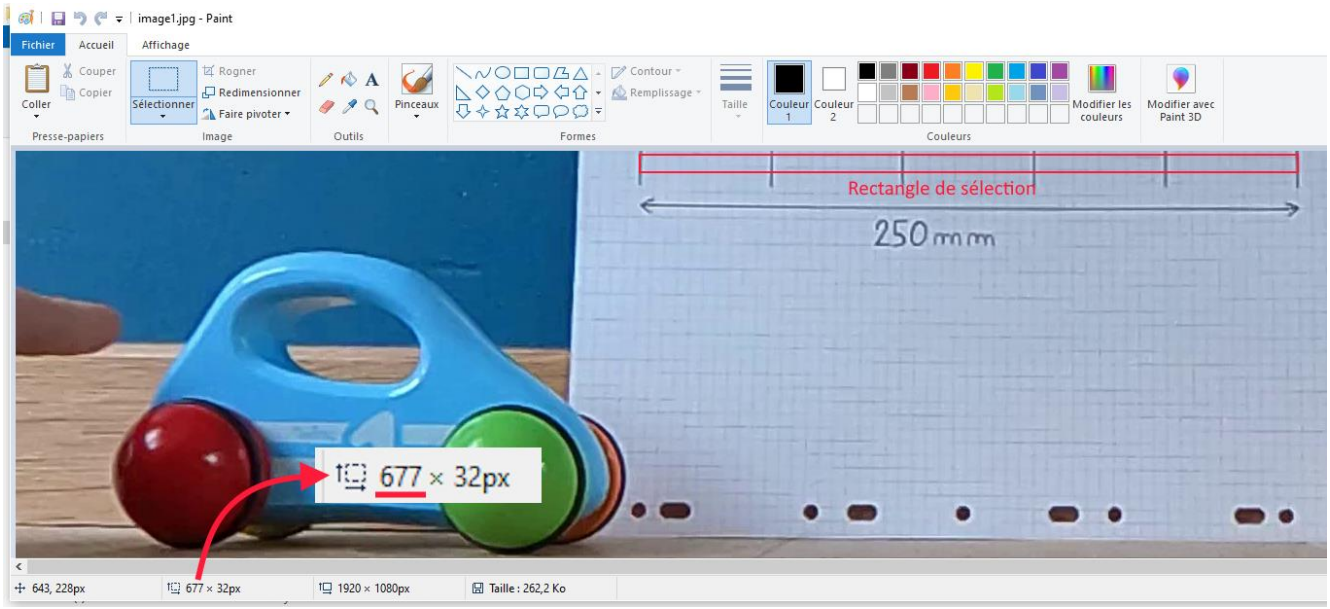
Mais globalement les conditions de lumière et de contraste sont satisfaisantes.

Remarque 1 : Ce TP a également été testé sur une vidéo présentant la décélération d'une trottinette filmée dans la cour du lycée avec un smartphone (image ci-dessous). Le tracking fonctionne également, mais selon le ROI choisi, le faible contraste avec les arbres de l'arrière-plan peut poser problème.

Remarque 2 : La vidéo proposée a été compressée pour être plus facilement manipulable et rendre le calcul de tracking plus rapide. Un service gratuit en ligne (clideo.com) a été utilisé pour cela. Attention aux encodages lors de la compression du fichier, qui peuvent rendre la vidéo illisible par openCV.

ACTIVITE 2 – TRAITEMENT DES DONNEES DE TRACKING

- Ouvrir le fichier "image1.jpg" issu du programme précédent dans Paint ou dans un autre éditeur d'images.
- Mesurer le nombre de pixels correspondant aux 250mm représentés sur l'image. Pour cela, on peut par exemple tracer une zone de sélection rectangulaire de longueur correspondant aux 250mm et observer ses dimensions en pixels en bas à gauche de l'écran.



Remarque : On néglige ainsi l'erreur induite par le fait que la référence n'est pas tout à fait dans le même plan que la voiture.

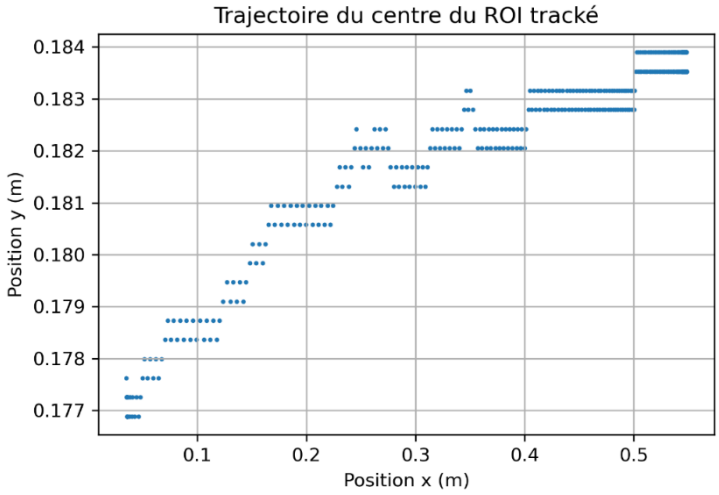
- On relève 677 pixels $\leftrightarrow$ 250 mm ce qui amène une variable $scale = 0.25/677$ qui vient multiplier les vecteurs $xtrack$ et $ytrack$.
- On modifie alors dans le fichier "2 – Traitement.py" les lignes 19, 20 et 21 comme suit :

```
17    # Conversion pixels -> m
18    # Mesure sur image 1 : 677 pixels = 250 mm
19    scale = 0.25/677
20    xtrack=xtrack*scale
21    ytrack=ytrack*scale
```

Remarque : On peut aussi intégrer le facteur d'échelle directement aux lignes 20 et 21. La variable $scale$ de la ligne 19 devient alors inutile (elle n'est pas utilisée ailleurs dans le code).

```
19 scale = 1 ← ligne inutile avec cette écriture
20 xtrack=(0.25/677)*xtrack
21 ytrack=(0.25/677)*ytrack
```

Q4 : *Courbe "0 – Trajectoire" - évolution pertinente ?*



La trajectoire est pertinente. L'échelle n'est pas orthonormée, mais en fait le déplacement sur y est très faible (7mm relevés). Le déplacement sur x, d'environ 500 mm est cohérent.

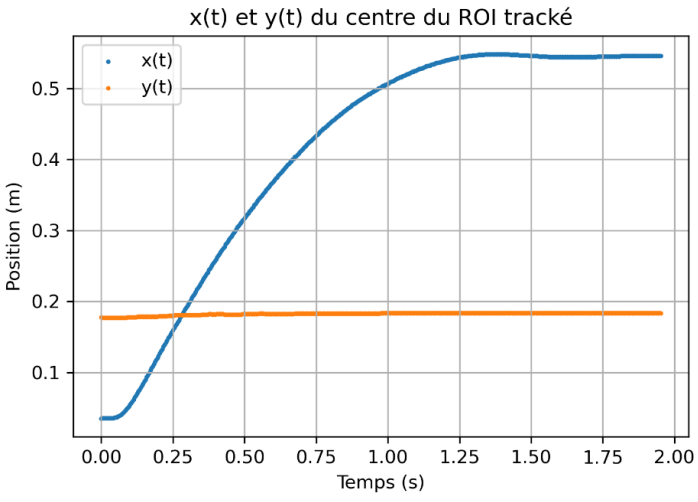
Quel phénomène observe-t-on sur l'axe y ?

On observe un effet de quantification, lié ici à la résolution de l'image.

Lien avec la caractéristique de la ligne 19 du script :

En se référant à l'échelle, 1 pixel correspond à 0,37mm (250 mm/677 pixels), valeur que l'on retrouve sur la courbe ci-dessus entre 2 "paliers" de quantification.

Q5 : *Courbe " 1 - Positions" - évolution pertinente au vu du mouvement de la voiture ?*



On remarque toujours un déplacement faible sur y.

Sur x, on note la mise en mouvement, puis l'augmentation de la variable de position jusqu'à stabilisation qui correspond à l'arrêt de la voiture.

Estimez-vous que les déplacements sur y doivent être pris en compte dans la suite de l'analyse ?

Non, ils paraissent négligeables face à ceux sur x.

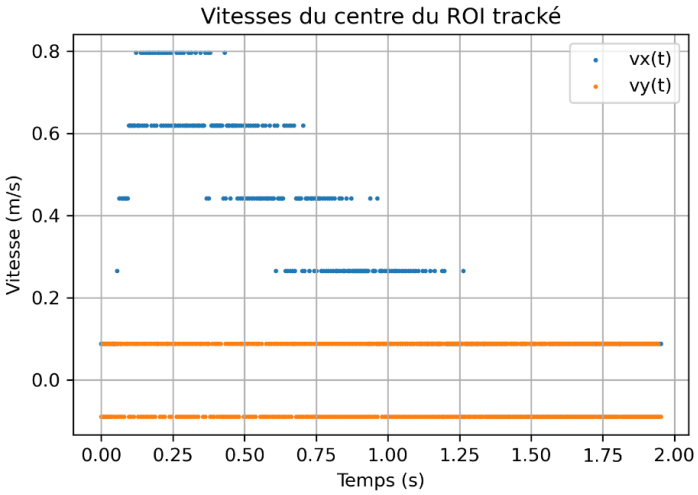
Q6 : *Commenter <u>et expliquer</u> l'évolution des vitesses obtenue sur la courbe "3 – Vitesses". Inclure les mots "résolution" et "échantillonnage" !*

On met ici en œuvre une dérivée numérique du type Euler implicite qui s'écrit (ligne 76) comme :

$$dx.append((x[i+1]-x[i])/(t[i+1]-t[i]))$$

Pour rappel, ce schéma se rapporte à la définition d'une dérivée : $\frac{d\,x(t)}{dt} = \frac{variation\ de\ x\ entre\ 2\ images}{variation\ de\ t\ entre\ 2\ images}$

Le tracé de la courbe de vitesses associé donne :



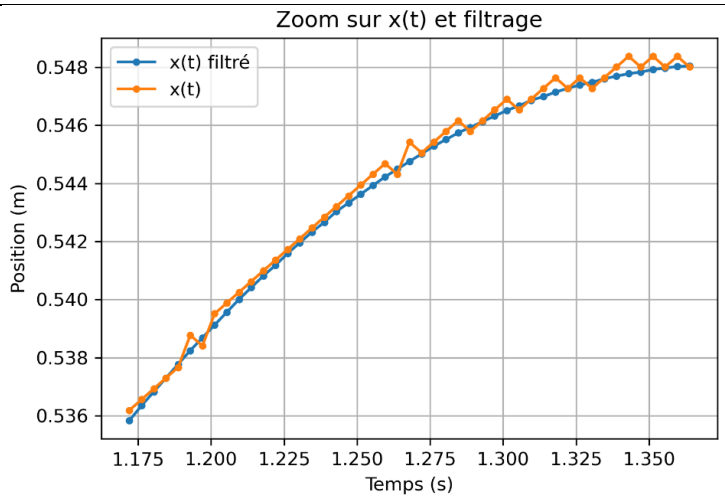
On remarque un "seuillage" de la vitesse qui ne permet pas son exploitation. En analysant la dérivation numérique $\frac{dx(t)}{dt}$*, on remarque que la variation de position entre 2 images correspond au maximum à 5 pixels alors que l'incrément de temps (période d'échantillonnage) est faible et constante.*

On peut dire que la résolution de l'image est faible par rapport à la fréquence d'échantillonnage de 240 images/s utilisée, pour permettre un calcul correct d'une dérivée numérique.

Pour que cela fonctionne mieux, on pourrait envisager de réduire la fréquence d'échantillonnage (framerate) tout en augmentant la résolution, ce qui est possible avec une caméra type GoPro.

Q7 : *Caractéristique retenue pour le filtre :*

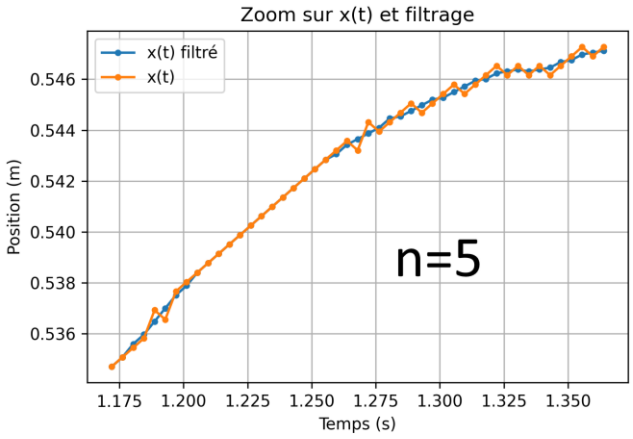
La solution pour contourner l'observation de la question 6 consiste à filtrer le signal de position avant dérivation. On choisit pour cela un filtre du type moyenne glissante, avec un nombre de valeurs (impair) de 21 qui paraît adapté au signal à filtrer :

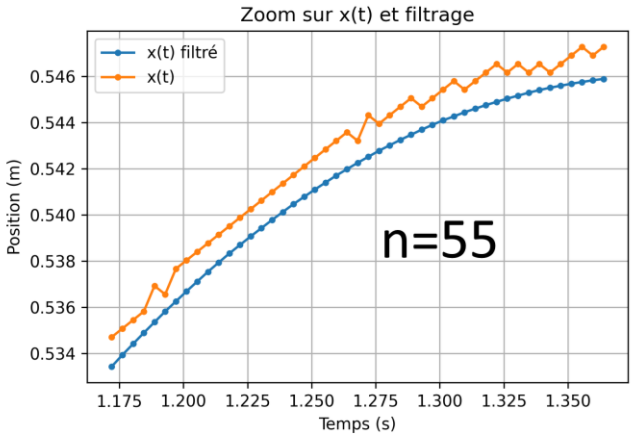


Les lignes 54 et 55 deviennent donc : $xtrackf = Filtre\_MG(xtrack, 21)$

$ytrackf = Filtre\_MG(ytrack, 21)$

Allure des courbes avec d'autres fenêtres de filtrage :





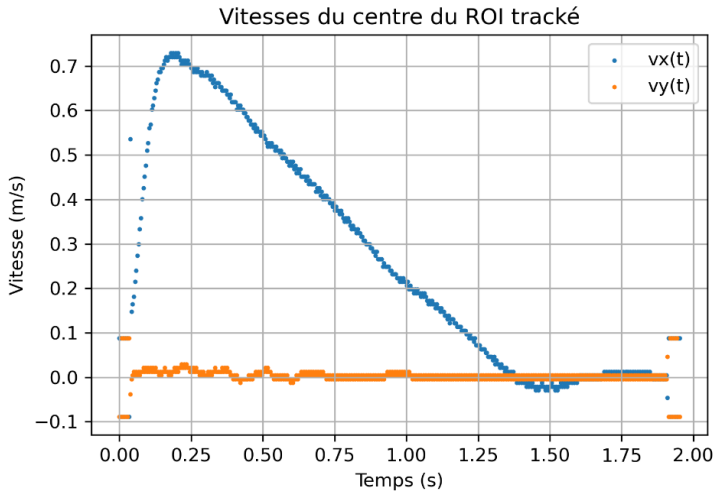
Avec n=5 (insuffisant), on note toujours des oscillations sur le signal en sortie de filtre. De plus, cette largeur de fenêtre de filtrage ne permet pas de gommer suffisamment les effets de quantification observés précédemment.

Avec n=55 (trop grand), le filtrage induit un décalage non souhaité de la courbe.

n=21 (par exemple) permet un bon compromis.

Courbe "3 – Vitesses" - résultats pertinents ?

Après réglage du filtre, la dérivée numérique obtenue devient bien plus exploitable qu'à la question 6 :



Début et fin de la décélération :

$t_d = 0{,}2\ s$ $t_f = 1{,}4\ s$

Remarque : Les lignes 101 à 105 du fichier "2 – Traitement.py" permettent de trouver les instants des images les plus proches des instants ci-dessus. En effet, les images ne sont pas acquises exactement en $t_d = 0{,}2\ s$ et $t_f = 1{,}4\ s$ mais en des valeurs légèrement différentes :

```
tps début décélération =  0.20437075592016835  s
tps fin décélération =  1.4055703009203302  s
```

ACTIVITE 3 : MODELISATION ET IDENTIFICATION D'UN MODELE

MODELISATION

Q8 : *Ecrire puis résoudre l'équation précédente dans le cas où $v \approx 0$.*

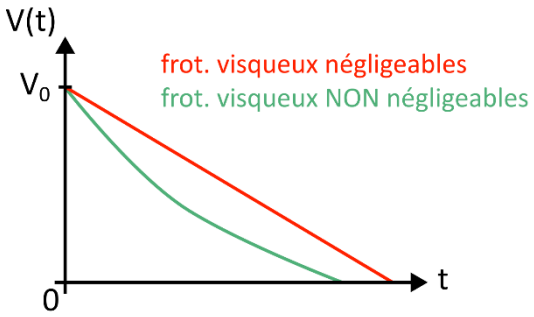
$$\dot{V}(t) = -\frac{F_f}{m} \rightarrow \quad V(t) = -\frac{F_f}{m}t + V_0$$

Allure de $V(t)$ dans ce cas :

L'équation de $V(t)$ correspond à une décroissance affine de la vitesse à partir de la vitesse V_0.

Allure de $V(t)$ si les frottements visqueux ne sont pas négligeables :

L'équation différentielle est alors du 1er ordre avec 2nd membre. L'évolution de $V(t)$ est alors non linéaire à partir de la vitesse V_0. En conservant le même terme de frottement sec F_f, on atteint la vitesse nulle plus rapidement.



Q9 : *Les frottements visqueux doivent-ils être pris en compte pour l'étude de la décélération de la voiture ?*

Non, car au vu de l'évolution de vitesse obtenue en question 7, on peut modéliser la décroissance de vitesse par une fonction affine.

Quels termes du modèle pourront être identifiés à partir des données expérimentales de la question 7 ?

Ordonnée à t correspondant au début de la décélération $= V_0$. Attention, il ne s'agit pas de l'ordonnée à l'origine ($t = 0$) car le début de la décélération est ici en $t_d = 0{,}2\ s$.

Coefficient directeur $= -\frac{F_f}{m}$. Il faut connaître l'un des paramètres pour pouvoir déterminer l'autre.

IDENTIFICATION DU MODELE AVEC SGDREGRESSOR

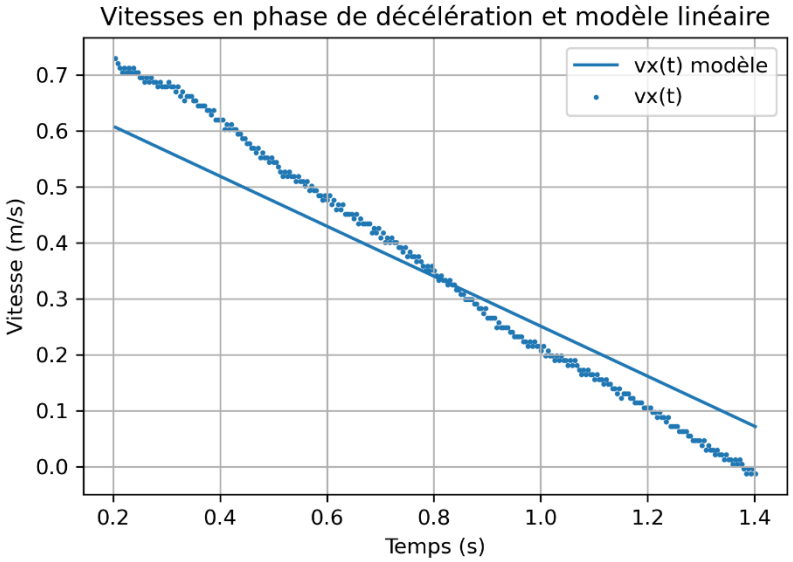
```
14
15  ####################   IDENTIFICATION MODELE LINEAIRE   ##########################
16
17  from sklearn.linear_model import SGDRegressor
18  tdd = np.asarray(td)
19  tdd = tdd.reshape(len(tdd), 1)     # transformation en un dataset (1 seul attribut)
20  model=SGDRegressor(max_iter=10000,tol=1e-4) # LIGNE A MODIFIER - création du modèle linéa
21  model.fit(tdd,vxd)      # entraînement du modèle
22  vxd_predit=model.predict(tdd)    # prédiction sur X
23  print("coef. directeur =",model.coef_) # affiche l'éq
24  print("ordonnée à l'origine =",model.intercept_)
25  print("coefficient de détermination R² = ",model.score(tdd,vxd))
```

Mise en forme des données

Création et entrainement du modèle

Utilisation du modèle pour effectuer des prédictions

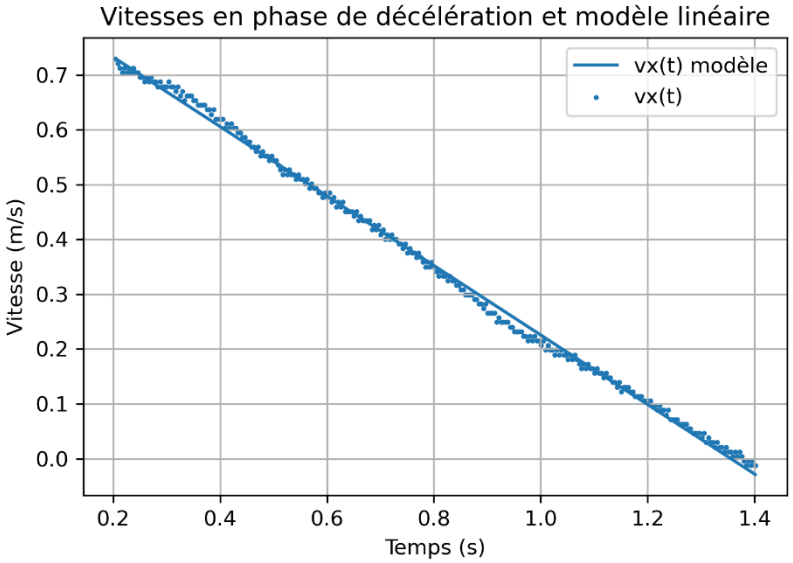
Q10 : *Pourquoi l'identification avec $SGDRegressor$ n'est-elle pas satisfaisante ?*



L'identification effectuée n'est pas satisfaisante à cause de la tolérance demandée à la régression qui n'est pas assez faible ($tol = 1e-4$).

Modification proposée :

On propose de réduire cette tolérance. On note que $tol = 1e-8$ donne une identification bien meilleure :



Q11 : *Valeurs des coefficients identifiés par le modèle :*

L'identification renvoie : coef. directeur = -0,634 – ordonnée à l'origine = 0,859

<u>Remarque :</u> En exécutant plusieurs fois le même code, les coefficients obtenus varient très légèrement. Ceci est dû à l'utilisation de l'algorithme SGD (Stochastic Gradient Descent) qui fait appel à une fonction aléatoire dans le calcul.

Evaluation des frottements secs F_f et la vitesse initiale V_0 :

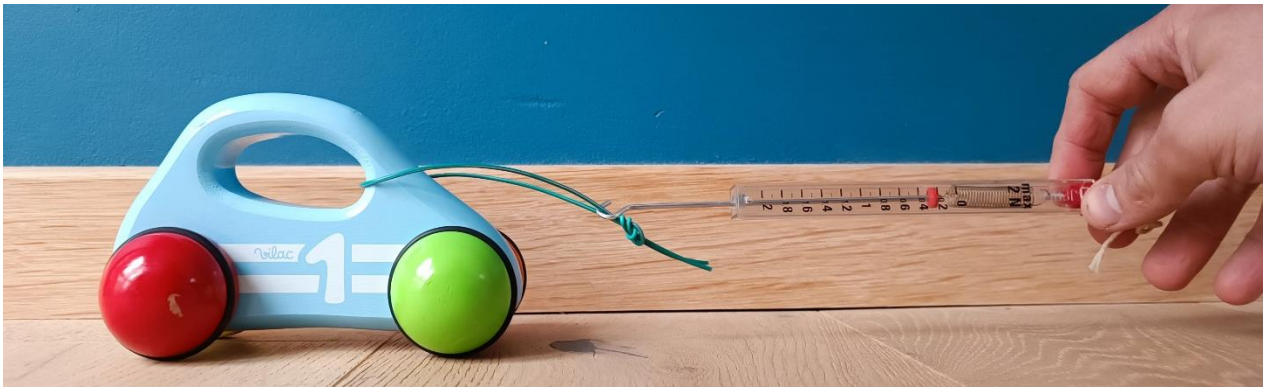
A partir de l'identification, et connaissant la masse $m = 0{,}281kg$, on a $-\frac{F_f}{m} = -0{,}634 \rightarrow F_f = 0{,}178\ N$

La vitesse initiale s'obtient avec $V(t = t_d) = -0{,}634 \times t_d + 0{,}86 = 0{,}73\ m/s$

VALIDATION EXPERIMENTALE DU PARAMETRE IDENTIFIE

Q12 : *Protocole expérimental simple permettant d'évaluer les frottements secs F_f :*

Les frottements secs peuvent être évalués au démarrage de la voiture (à partir d'une situation statique). On peut pour cela chercher quelle est la force de traction nécessaire à la mise en mouvement de la voiture. Concrètement, on peut utiliser un dynamomètre accroché à la voiture, sur lequel on vient tirer pour relever la force nécessaire à la mise en mouvement :



L'effort relevé étant très faible, cette mesure peut être un peu délicate dans le cas de la voiture, d'où la valeur de précision pas vraiment négligeable donnée à la question suivante. Selon le cas d'étude traité (par exemple la décélération de la trottinette évoquée en page 2 de ce corrigé), cet effort sera plus ou moins facile à mesurer avec un dynamomètre.

On obtient environ $F_f = 0{,}2N$.

Q13 : Expérimentalement, $F_f \approx 0{,}2\ N$ *(précision de la mesure évaluée à* $\pm 0{,}03\ N$*). Comparer cette valeur à celle obtenue à la question 11 :*

$$F_{f\ identifie} = 0{,}178\ N \quad \leftrightarrow \quad F_{f\ mesure} = 0{,}2\ \pm 0{,}03\ N$$

En prenant en compte la tolérance sur la mesure (précision), les deux valeurs mesurées sont en accord.

Q14 : *Commenter la modélisation. Voyez-vous des éléments qui ont été négligés ?*

On a considéré dans la modélisation que les frottements visqueux étaient négligeables, ce qui semble pertinent au vu de l'évolution linéaire de la courbe de vitesse.

Par ailleurs, on a considéré que la voiture se résumait à un simple solide en translation, ce qui n'est pas vrai en réalité car les roues tournent. L'énergie cinétique, limitée à celle d'une masse m en mouvement de translation dans le modèle, comporte en réalité d'autres termes associés à la rotation des deux essieux qui a été négligée.

Prendre en compte ces inerties reviendrait à chercher une masse équivalente à l'ensemble (notion d'inertie équivalente ramenée à un solide en translation). Cette masse équivalente étant plus grande que m, ceci amènerait une nouvelle valeur de F_f un peu plus élevée.

IDENTIFICATION D'UN MODELE DU TYPE "RESEAU DE NEURONES"

On repère les différentes parties associés au réseau de neurones dans le code ci-dessous :

```
17  ######################   UTILISATION D'UN RESEAU DE NEURONES   ############################
18
19  from sklearn.neural_network import MLPRegressor # Multi Layer Perceptron Regressor
20  from sklearn.model_selection import train_test_split
21  tdd_train, tdd_test, vxd_train, vxd_test = train_test_split(tdd, vxd,test_size=0.33,random_state=1)
22  regr = MLPRegressor(random_state=1, max_iter=100000,hidden_layer_sizes = (2),activation='logistic',tol=1e-12).fit(tdd_train, vxd_train)
23  vxd_predict=regr.predict(tdd_test)
24  print('coefficient R² : ',regr.score(tdd_test, vxd_vxd_test))
25
```

Séparation données d'entrainement / de test

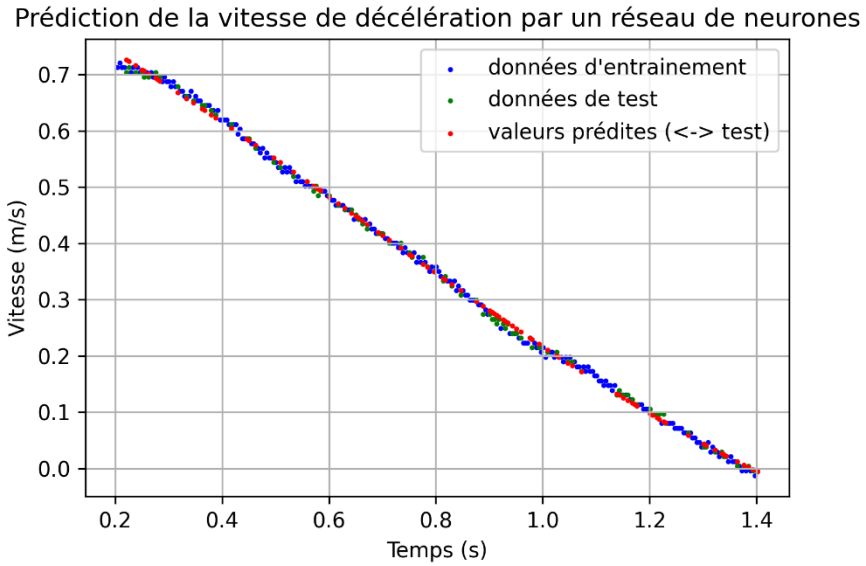
Création et entrainement du réseau de neurones

Fonction d'activation sigmoide

1 couche cachée comportant 2 neurones

Test du modèle sur le dataset de test

Remarque : On a ici une seule couche cachée comportant 2 neurones. Si on souhaitait par exemple une première couche cachée de 2 neurones puis une seconde couche cachée de 4 neurones, on écrirait ci-dessus $hidden\_layer\_sizes = (2,4)$. On ne s'intéressera pas ici au choix du nombre de couches et de neurones cachés.



Q15 : *Représenter le réseau de neurones créé par le script (agencement de plusieurs perceptrons) et y annoter les différents poids et biais extraits du modèle :*

Structure du réseau de neurones :

- 1 attribut d'entrée = le temps ➔ 1 neurone dans la couche d'entrée
- 1 attribut de sortie = la vitesse ➔ 1 neurone dans la couche de sortie
- 1 couche de 2 neurones cachés (choix lors de la création du réseau de neurones)

Fonction d'activation = sigmoïde sur les 2 neurones de la couche cachée

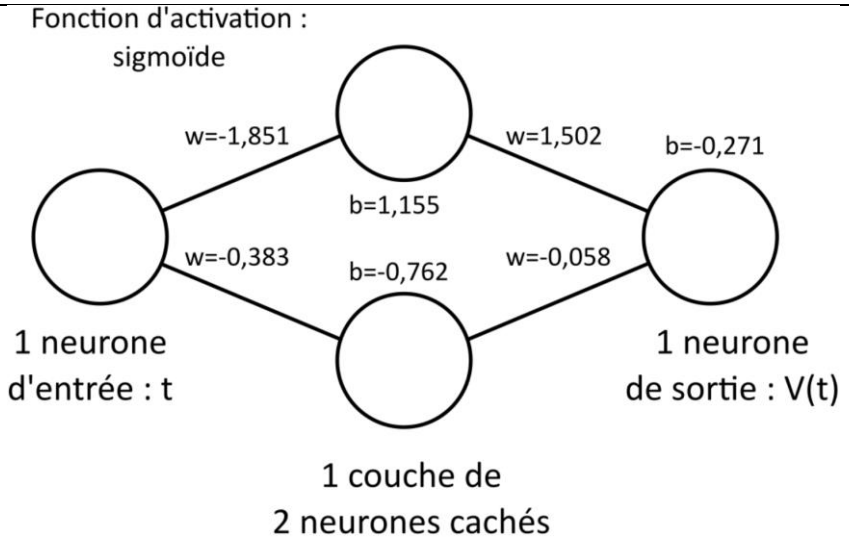
Les fonctions données permettent d'extraire les poids (w – regr.coefs) et les biais (b – regr.intercepts) des neurones du réseau de neurones. On rappelle que :

- Le neurone d'entrée ne possède aucun coefficient et sert juste d'entrée au réseau.
- Le neurone de sortie ne possède pas de fonction d'activation dans le cas d'un problème de régression (ce qui est le cas ici). Sa sortie correspond directement à son excitabilité $z = \sum_i w_i . x_i + b$, x_i étant les entrées du neurone (correspondant aux sorties de tous les neurones de la couche précédente).

regr.coefs_ ➔ [array([[-1.8513646 , -0.38303323]]),array([[1.50297306], [-0.05856014]])]

regr.intercepts_ ➔ [array([1.15554711, -0.76253507]), array([-0.27159376])]

A partir de ces éléments, on peut construire le réseau de neurones suivant :



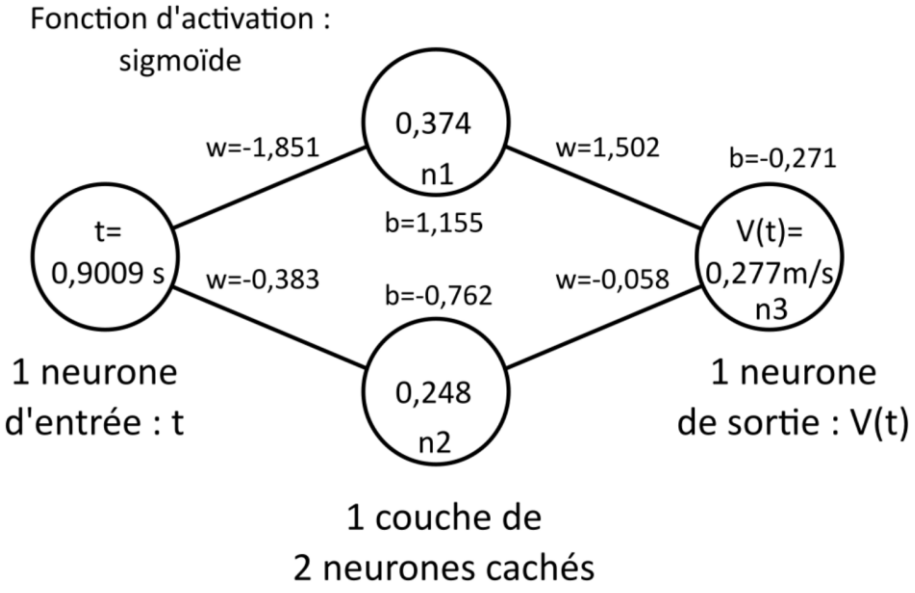
Q16 : *Calculant manuellement la réponse du réseau à un échantillon du dataset de test :*

Recherche d'une valeur de temps proche de $t = 0{,}9\ s$:
Pour aller chercher une valeur proche de $t = 0{,}9\ s$, double cliquer sur la variable tdd_test depuis l'explorateur de variables, puis chercher l'indice $i\ = 71$) de la valeur proche de 0,9. Les valeurs sont reclassées dans un ordre aléatoire, ce qui ne facilite pas la recherche. On peut trouver l'indice i en copiant les données depuis l'explorateur de variable vers un tableur et en utilisant une fonction de recherche pour trouver la valeur 0,9009 s.

Recherche de la valeur de vitesse mesurée (réelle) associée à l'indice i :
On cherche la valeur de vitesse mesurée correspondante en relevant la valeur de la variable vxd_test à l'indice i. Ici $t = 0{,}9009\ s$ ➔ $V(t_i)_{réelle} = 0{,}274\ m/s$.

Recherche de la valeur de vitesse prédite par le réseau de neurones associée à l'indice i :
On cherche la valeur de vitesse mesurée correspondante en relevant la valeur de la variable $vxd\_predict$ à l'indice i. Ici $t = 0{,}9009\ s$ ➔ $V(t_i)_{predite} = 0{,}277\ m/s$.

Retrouvons cette valeur prédite par le réseau de neurones par le calcul, pour comprendre son fonctionnement :



Détail du calcul de la sortie de chaque neurone du réseau :

Neurone n1 : Excitabilité $z_1 = -1{,}851 \times 0{,}9009 + 1{,}155 = -0{,}512$

Fonction sigmoïde : $\Phi_{s1}(z_1) = \frac{1}{1+e^{-z_1}} = \frac{1}{1+e^{0{,}512}} = 0{,}374$

Neurone n2 : Excitabilité $z_2 = -0{,}383 \times 0{,}9009 - 0{,}762 = -1{,}107$

Fonction sigmoïde : $\Phi_{s2}(z_2) = \frac{1}{1+e^{-z_2}} = \frac{1}{1+e^{1{,}107}} = 0{,}248$

Neurone n3 : Excitabilité $z_3 = 1{,}502 \times \Phi_{s1}(z_1) - 0{,}058 \times \Phi_{s2}(z_2) - 0{,}271 = 0{,}277\ m/s$

Pas de fonction d'activation pour le neurone de sortie si problème de régression.

On retrouve bien la valeur $V(t_i)_{predite} = 0{,}277\ m/s$ *relevée dans la variable* $vxd\_predict$.

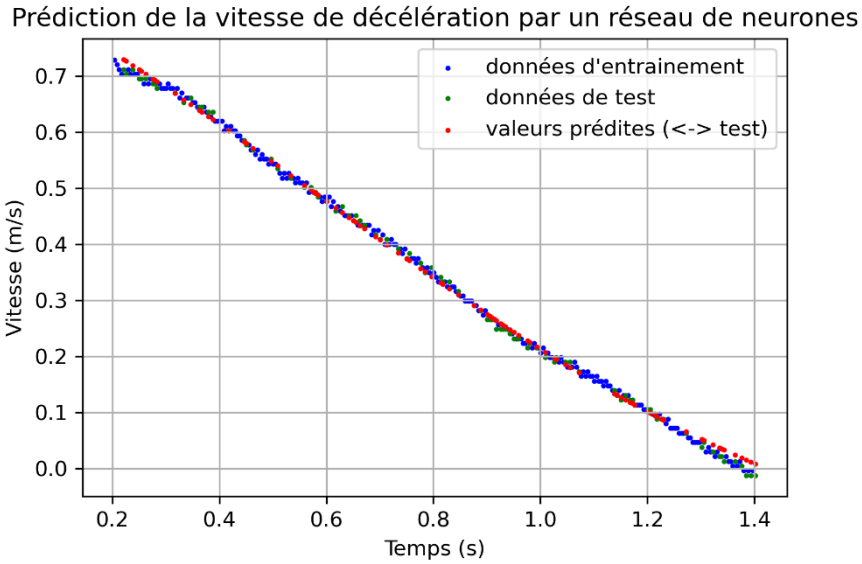
VARIATION DES PARAMETRES DU RESEAU DE NEURONES :

Modification du nombre de couches cachées et du nombre de neurones par couche cachée :

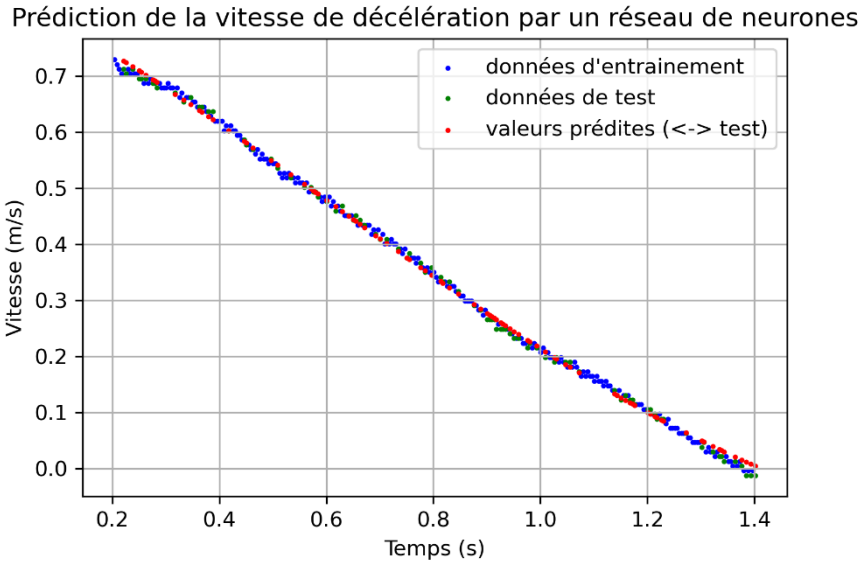
```
22    regr = MLPRegressor(random_state=1, max_iter=100000,hidden_layer_sizes = (4,4),
                          activation='logistic',tol=1e-8).fit(tdd_train, vxd_train)
```

2 couches cachées de 4 neurones chacune

1 couche de 2 neurones cachés – tolérance = 10^{-8} (réseau traité aux questions 15 et 16) :



2 couches de 4 neurones cachés – tolérance = 10^{-8} :



➔ On obtient un résultat très légèrement meilleur qu'avec 2 neurones cachés.

Remarque :

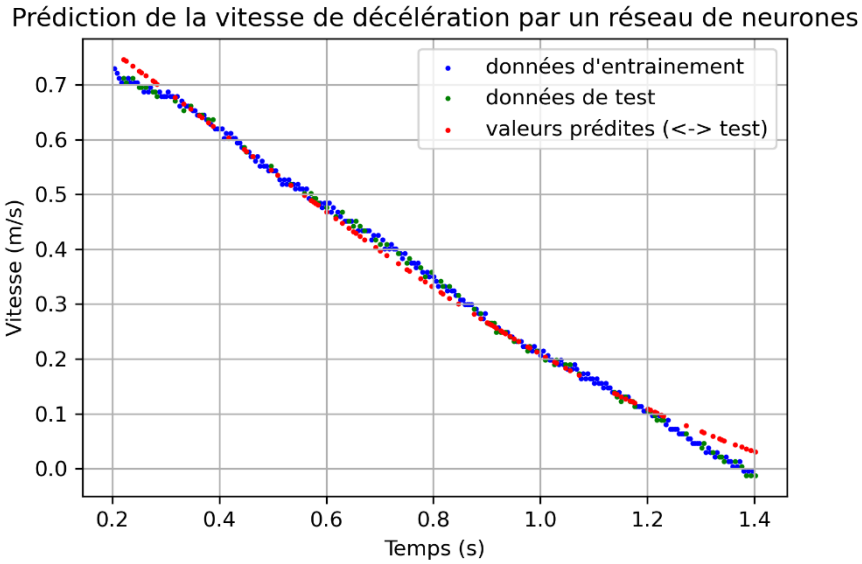
L'apport de neurones cachés (sous forme de nouvelles couches ou d'un plus grand nombre de neurones par couche cachée) apporte de la complexité au modèle. Le modèle peut alors mieux identifier des problèmes de régression complexes. Ici on travaille sur un problème de régression linéaire très simple. Il est donc normal que l'apport de neurones cachés supplémentaires n'apporte pas grand-chose à la précision de l'identification.

Modification de la tolérance du solveur :

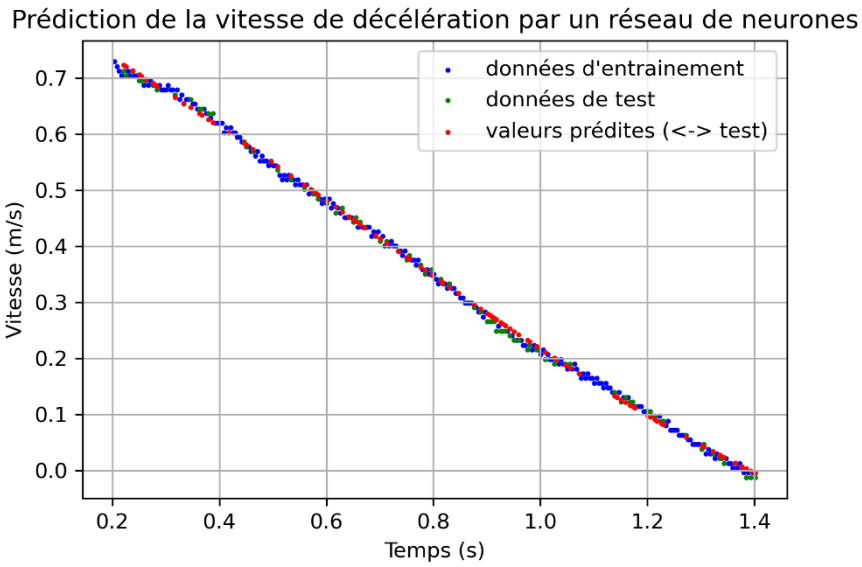
```
22    regr = MLPRegressor(random_state=1, max_iter=100000,hidden_layer_sizes = (2),
          activation='logistic',tol=1e-12).fit(tdd_train, vxd_train)
```

Valeur de tolérance à modifier

1 couche de 2 neurones cachés – tolérance = 10^{-7} :



1 couche de 2 neurones cachés – tolérance = 10^{-12} :



➔ La modification de la tolérance a une influence très importante sur l'identification du modèle.

Remarque :

La valeur de tolérance spécifie à quel moment l'itération permettant la détermination des poids w et biais b optimaux doit se terminer. Il est donc normal qu'elle influe beaucoup sur la qualité de l'identification. On note que diminuer la tolérance augmente nettement le temps de calcul qui reste cependant faible car le problème est simple (peu de coefficients à identifier).

Pour information OpenAI GPT-4 possède 1,7 trilliards de paramètres w et b. Ici on en a 7 !

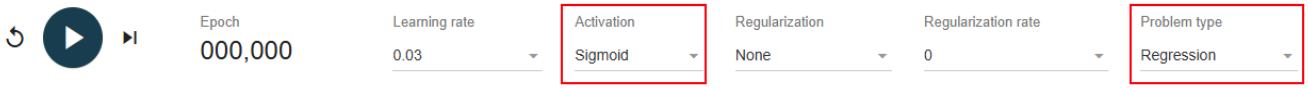
PLAYGROUND TENSORFLOW ➔ INFLUENCE DES NEURONES DES COUCHES CACHEES

Se rendre à l'adresse suivante pour la version complète du simulateur :

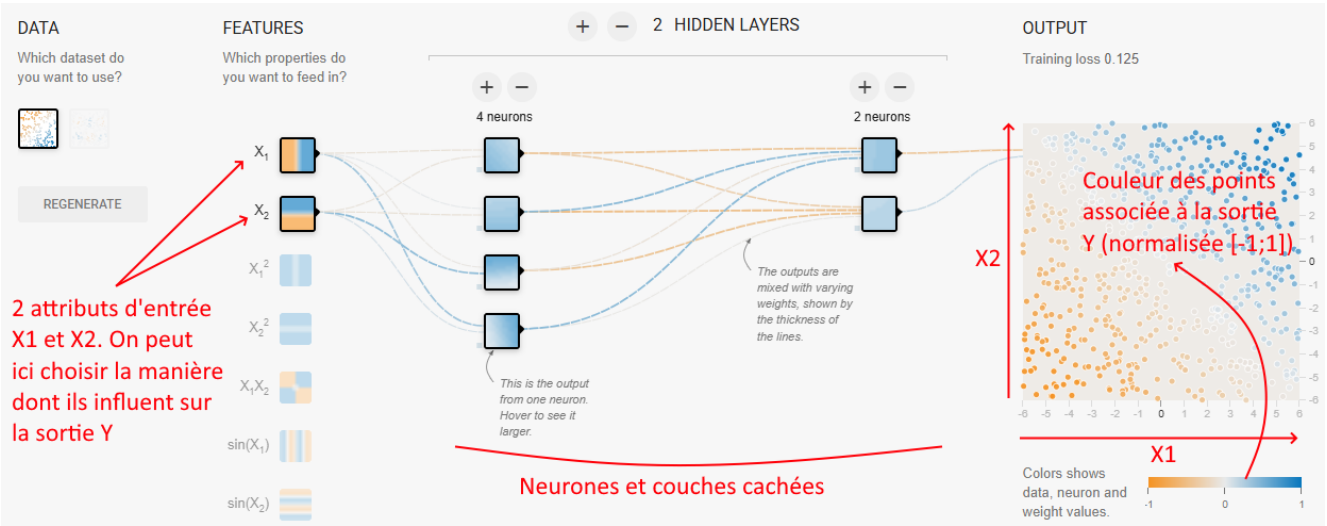
https://playground.tensorflow.org/ ou à cette adresse pour la version simplifiée pour cet exemple.

Remarque : dans l'analyse ci-dessous, on effectue une analyse qualitative de la forme des données du dataset. On n'attendra pas de résultats quantitatifs puisque les données de sortie Y sont normalisées entre -1 et 1.

- Dans la barre de menu du haut, choisir un problème de régression et une fonction d'activation de type sigmoïde, comme cela a été traité dans le début de ce TP.



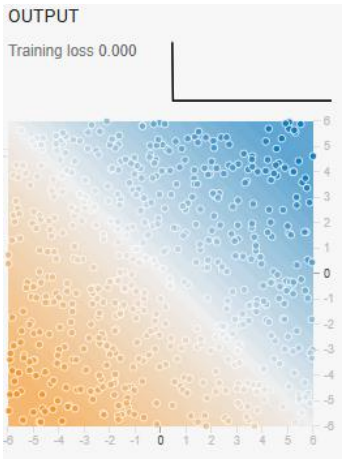
La partie inférieure de l'interface se présente par défaut comme suit :



On note que le problème de régression proposé peut avoir 2 attributs d'entrée (ou 1 si l'on n'en laisse qu'un seul en entrée).

CAS 1 : AUCUN NEURONE CACHE

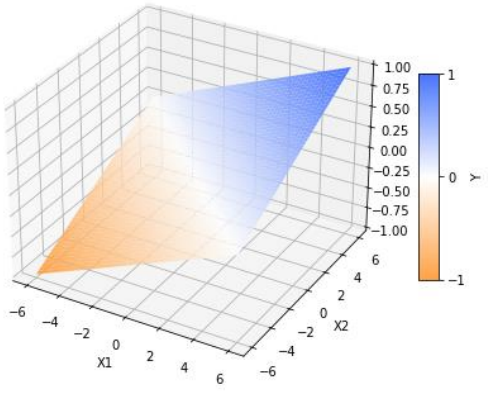
- Partons du réseau de neurones présenté sur le graphique ci-dessus : 2 neurones d'entrée associés aux attributs X1 et X2 – 1 couche cachée de 4 neurones – 1 couche cachée de 2 neurones – 1 neurone de sortie ➔ Sortie Y.
- Commençons par supprimer toutes les couches cachées : il ne reste alors plus que les 2 neurones d'entrée (qui ne réalisent aucune opération mathématique) et un neurone de sortie.



Rappel : On rappelle que dans un problème de régression, le neurone de sortie ne comporte pas de fonction d'activation. Il se résume donc à un calcul d'excitabilité (normalisée sur l'ensemble du dataset dans le cas présent).

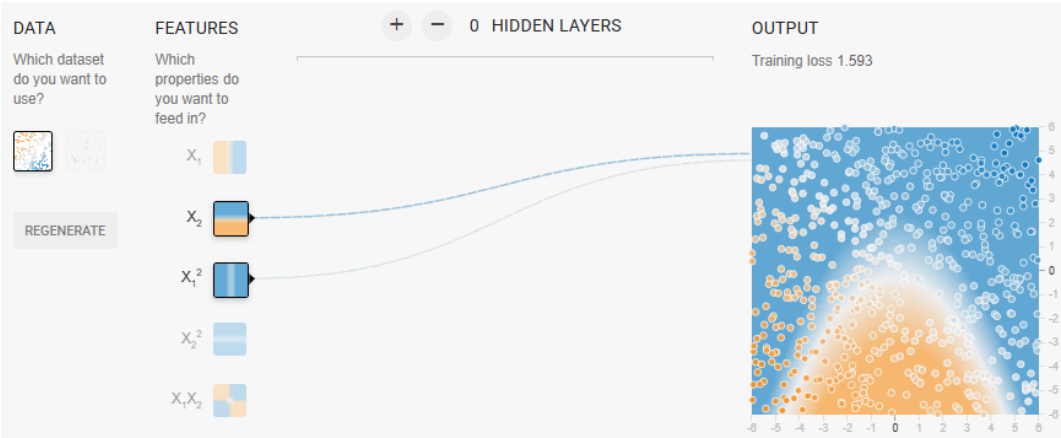
- Lancer l'entrainement du réseau de neurones en cliquant sur ▶ en haut à gauche. On voit très rapidement apparaître une carte colorée ayant une ligne Y=0 en diagonale, issue de l'influence linéaire choisie pour les deux attributs X1 et X2. En représentant cette carte sur un graphique en 3D, on obtient une surface plane (voir page suivante).

Explication : Pour l'instant, les attributs d'entrée X1 et X2 influencent tous deux la sortie Y de manière linéaire. Le réseau de neurones comporte 2 poids associés à chacune de ses entrées (+ 1 biais qui est nul dans cet exemple car on a normalisé les données sur un intervalle symétrique). On peut les obtenir en plaçant le curseur sur les liens. On note qu'ils sont identiques ce qui montre que les deux attributs influencent la sortie de la même manière.



On remarque ici que pour effectuer une régression linéaire, avoir un seul neurone est suffisant.

- Remplaçons maintenant l'influence linéaire de l'attribut X1 par une influence de forme $X1^2$:

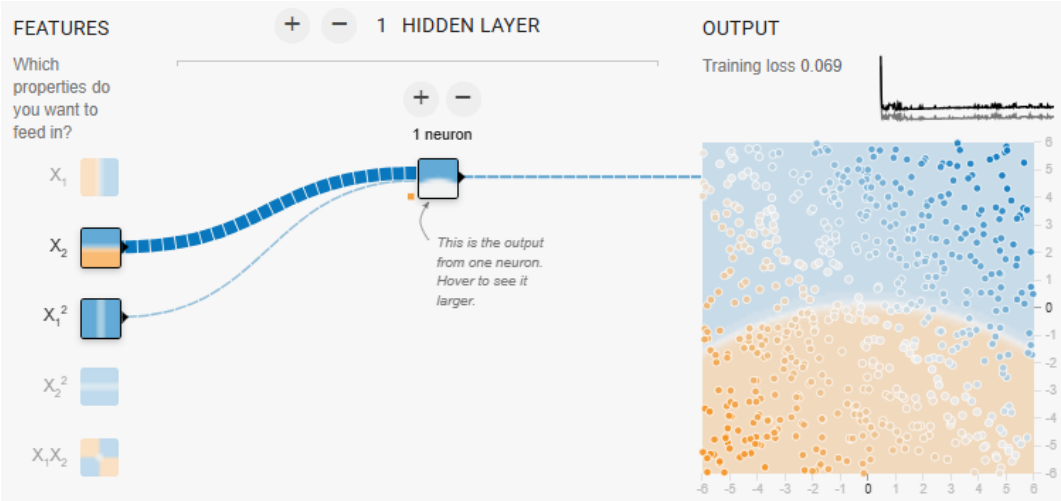


On remarque que le paramètre X2 influe toujours linéairement sur la sortie, mais que pour un X2 fixé, Y varie proportionnellement à $X1^2$.

- Lancer l'entrainement du réseau de neurones en cliquant sur ▶. On remarque que l'algorithme ne parvient pas à converger car le seul neurone de sortie ne permet pas de reproduire cette tendance non linéaire.

Cas 2 : Une couche cachee

- Conserver l'entrée précédente et ajouter une couche de 1 neurone caché.
- Relancer l'entrainement du réseau de neurones. On voit que cette fois, la présence d'un neurone caché avec sa fonction d'activation non linéaire permet de retrouver une non-linéarité dans la sortie :



On peut poursuivre les explorations avec cet outil, mais on comprend par l'exemple précédent que plus les données d'entrée ont des tendances complexes, plus il sera nécessaire d'augmenter le nombre de neurones et de couches cachées.