CONCEPTION D’UNE DYNAMO DE VÉLO

Gabriel CARSENAT
N° 29451

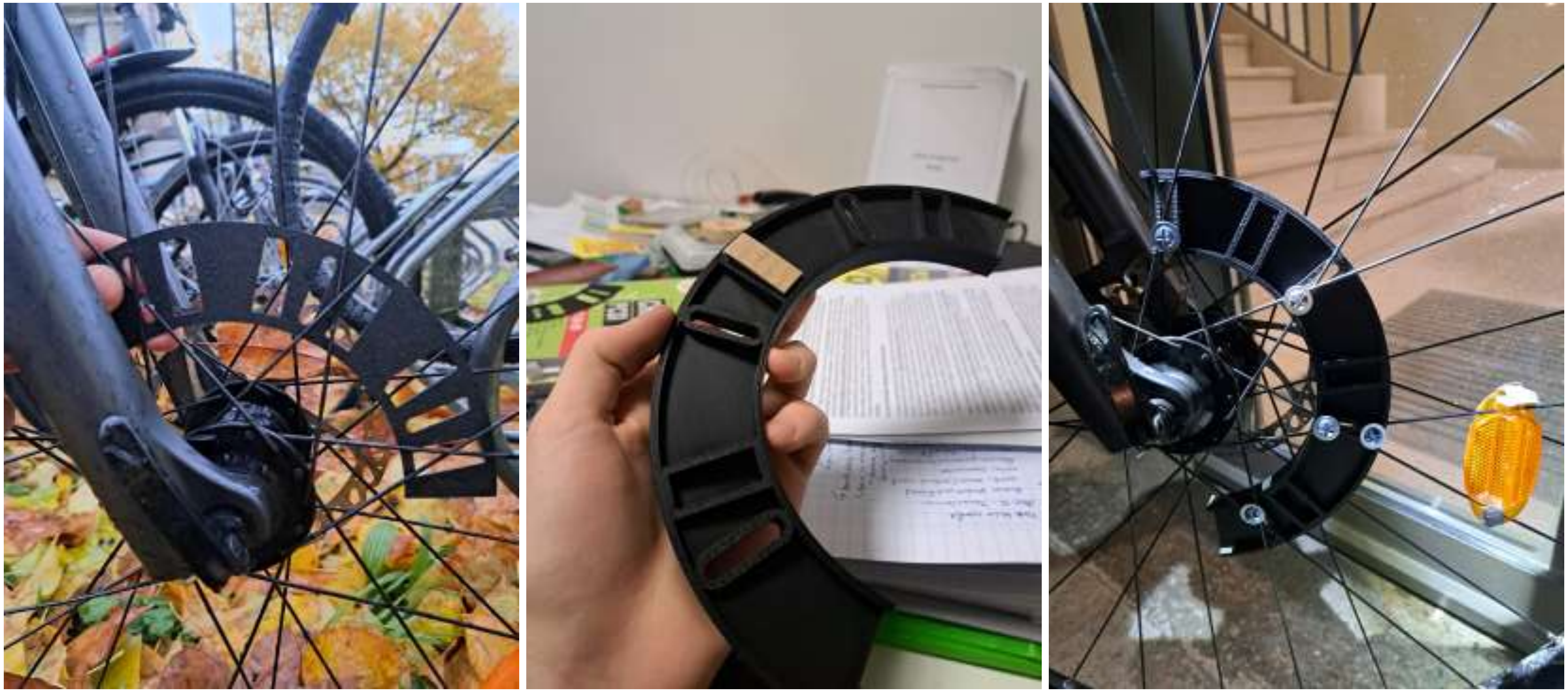
PROBLÉMATIQUE

- Dans quelle mesure la dynamo peut-elle se substituer aux piles et batteries comme source d'énergie électrique sur un vélo?

Plan:

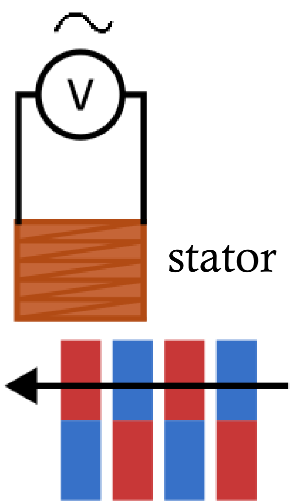
1. Fabrication du dispositif
2. Mesures d'amplitude
   1. Effet des bobines
   2. Effet de la vitesse
   3. Effet de la distance
3. Etude des aimants et simulation

FABRICATION ET MONTAGE (ROTOR)



Gêne minimale du dispositif de maintien

FONCTIONNEMENT



Rotor

(aimants disposés en cercle)

Loi de Faraday pour une bobine idéale:

$$e = -\frac{d\phi_{(s)}}{dt}$$

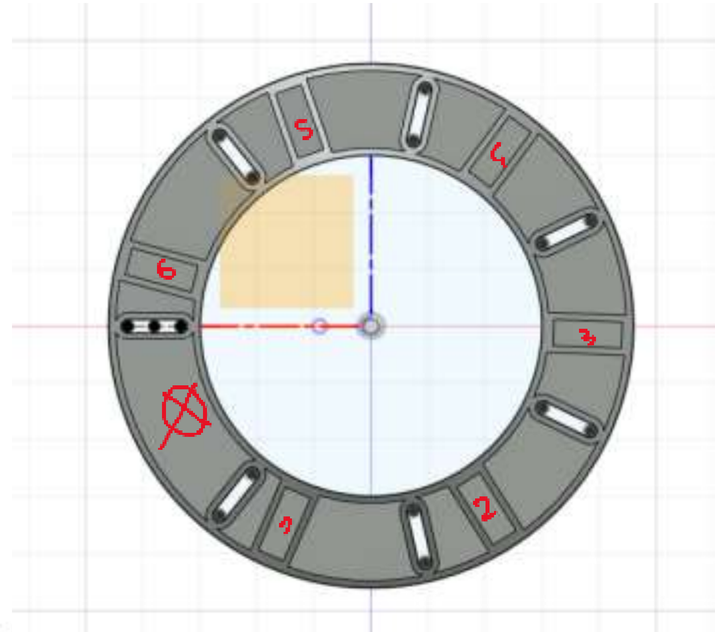
avec $\phi_{(s)} = N \iint_{(s)} \overrightarrow{B(M)}.\overrightarrow{dS} = N * \overline{B_n}\, S$

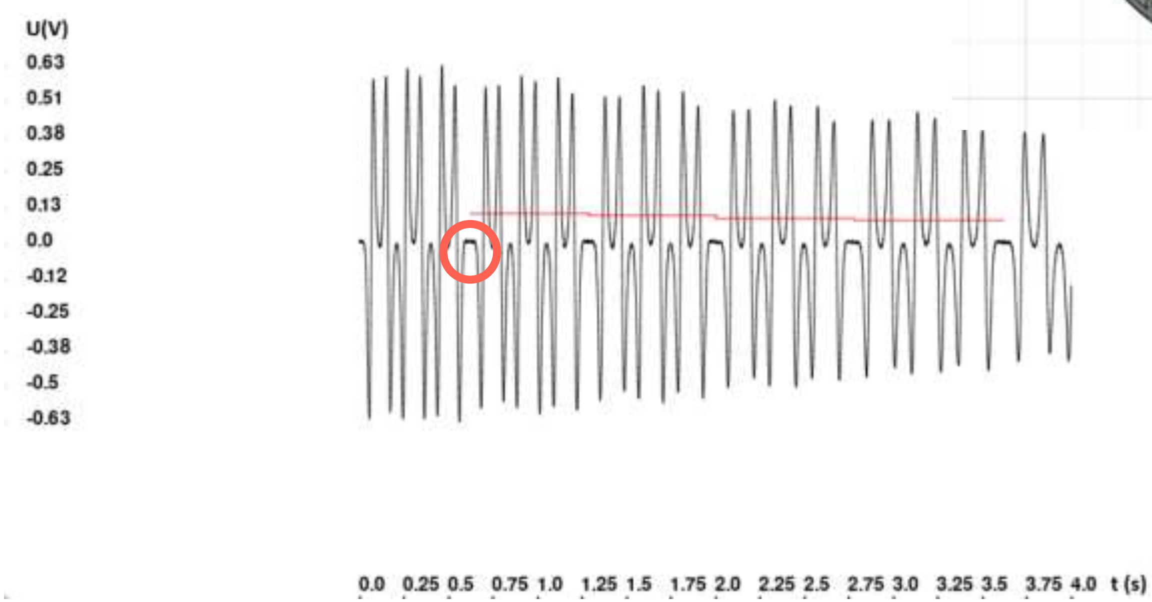
Influence:

- Distance car B(x,y,z, t)
- Dimension de la bobine
- Vitesse d'alternation

FABRICATION ET MONTAGE (BOBINES)

PROCÉDÉ DE MESURE ET TRAITEMENT

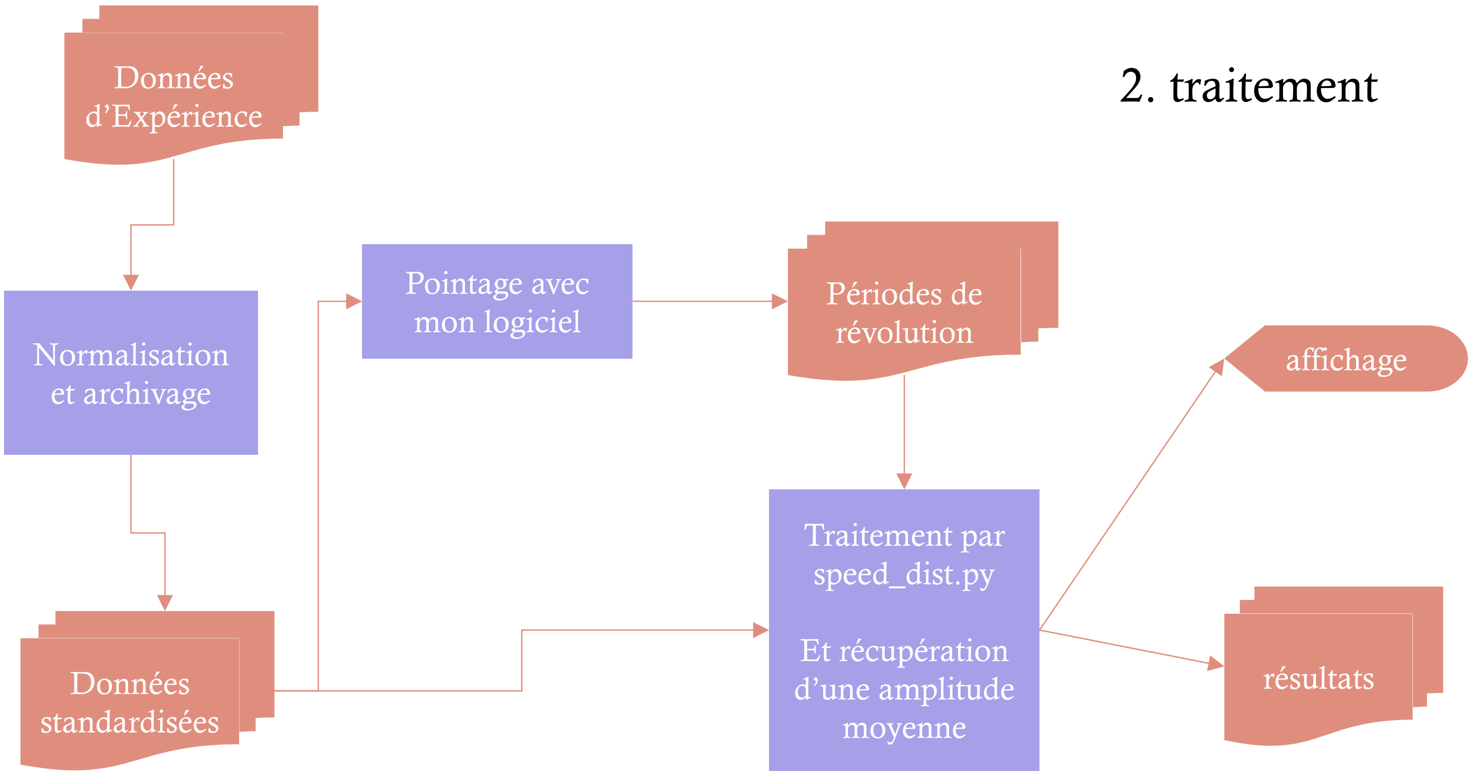




1: principe

TRAITEMENT DES SÉRIES TEMPORELLES

2. traitement

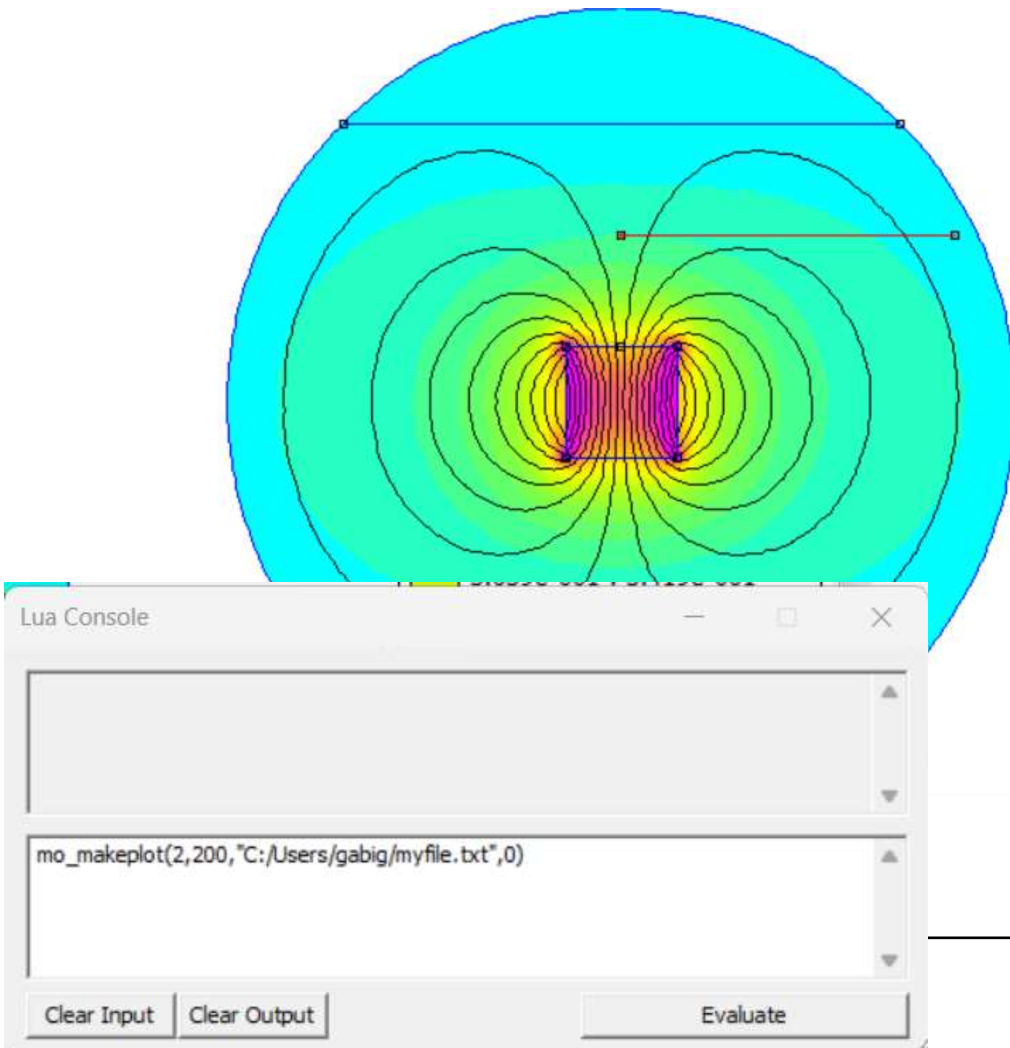


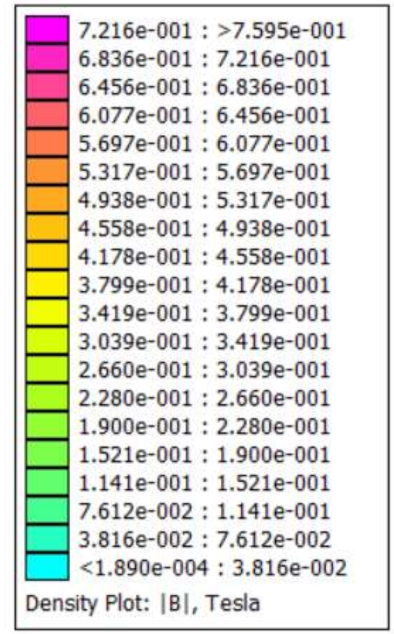
$$V_{eff}(T) = \sqrt{moy(V^2)_T} \approx \frac{1}{N}\sum_{i=1}^{N}|V_i(T)|$$

INFLUENCE DES BOBINES

CHAMP DE L'AIMANT

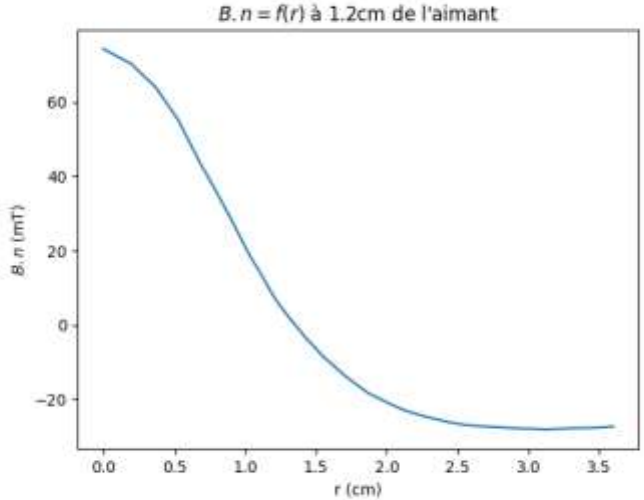
Réalisé sur le logiciel femm
(resolution par éléments finis).

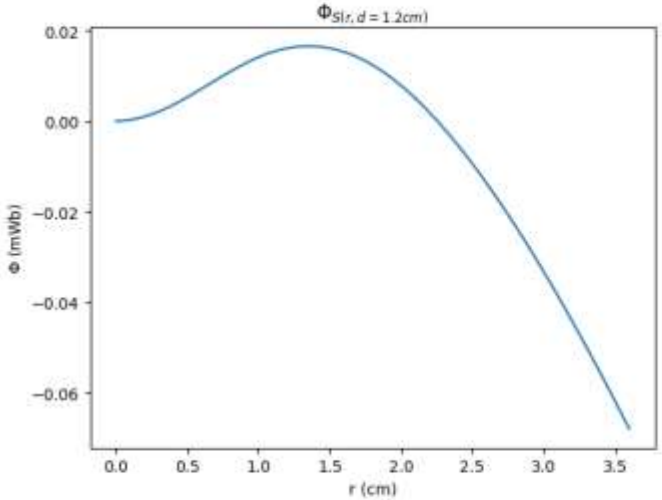


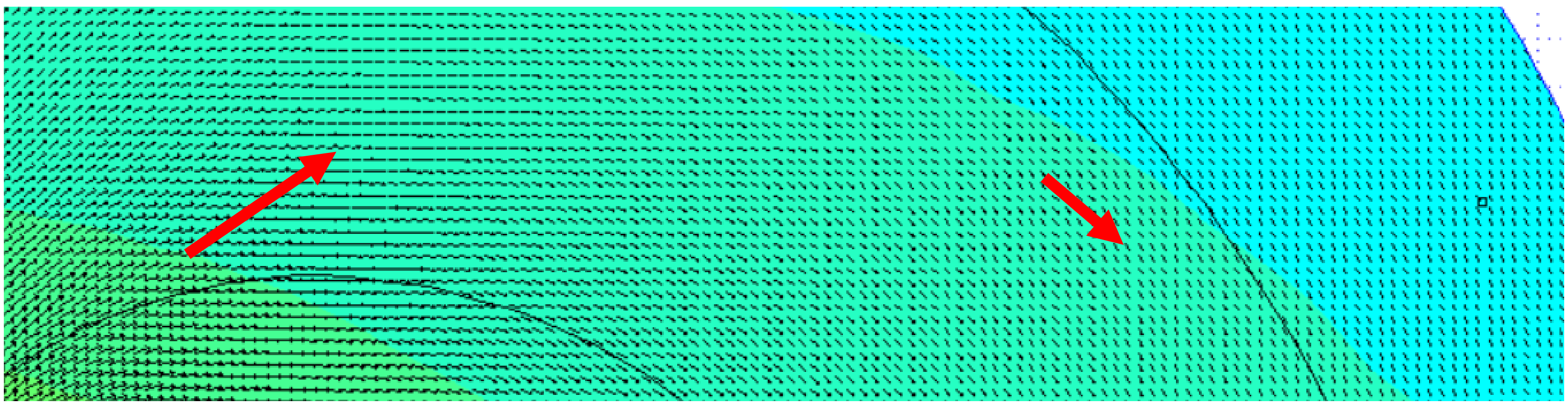


INFLUENCE DES BOBINES

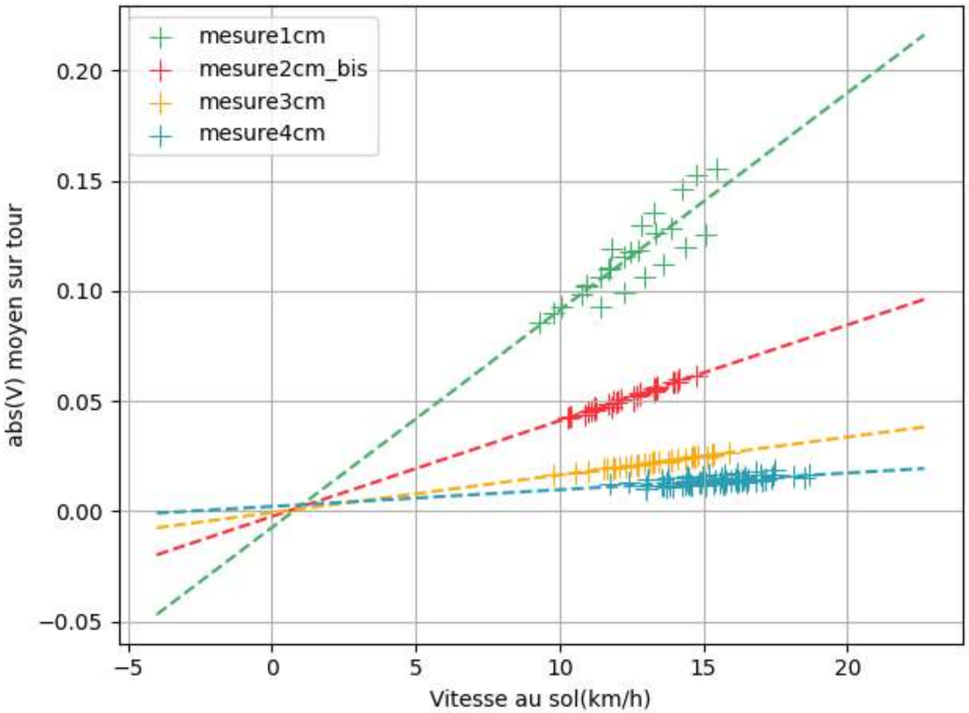
Réalisé avec le logiciel femm.
(fonction mo_makeplot)







EFFET DE LA VITESSE



Expérimentalement l'amplitude linéaire de la vitesse

B moyen.

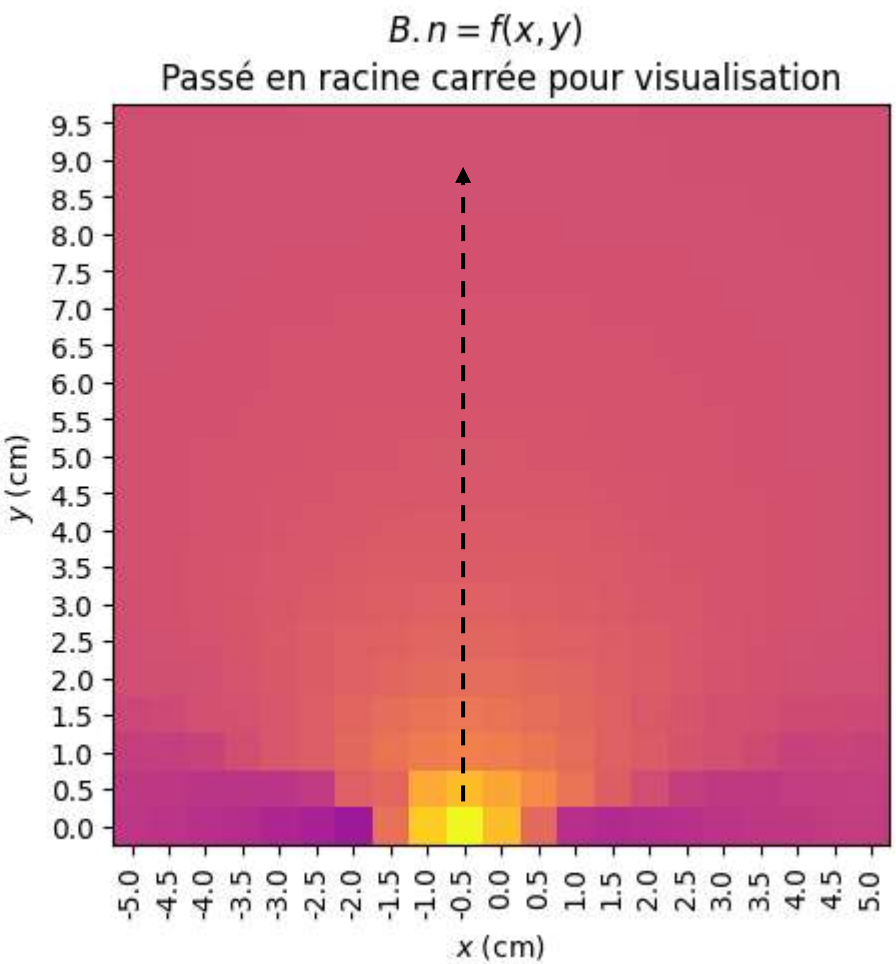
$$B_n = B_0 \sin(\omega t)$$

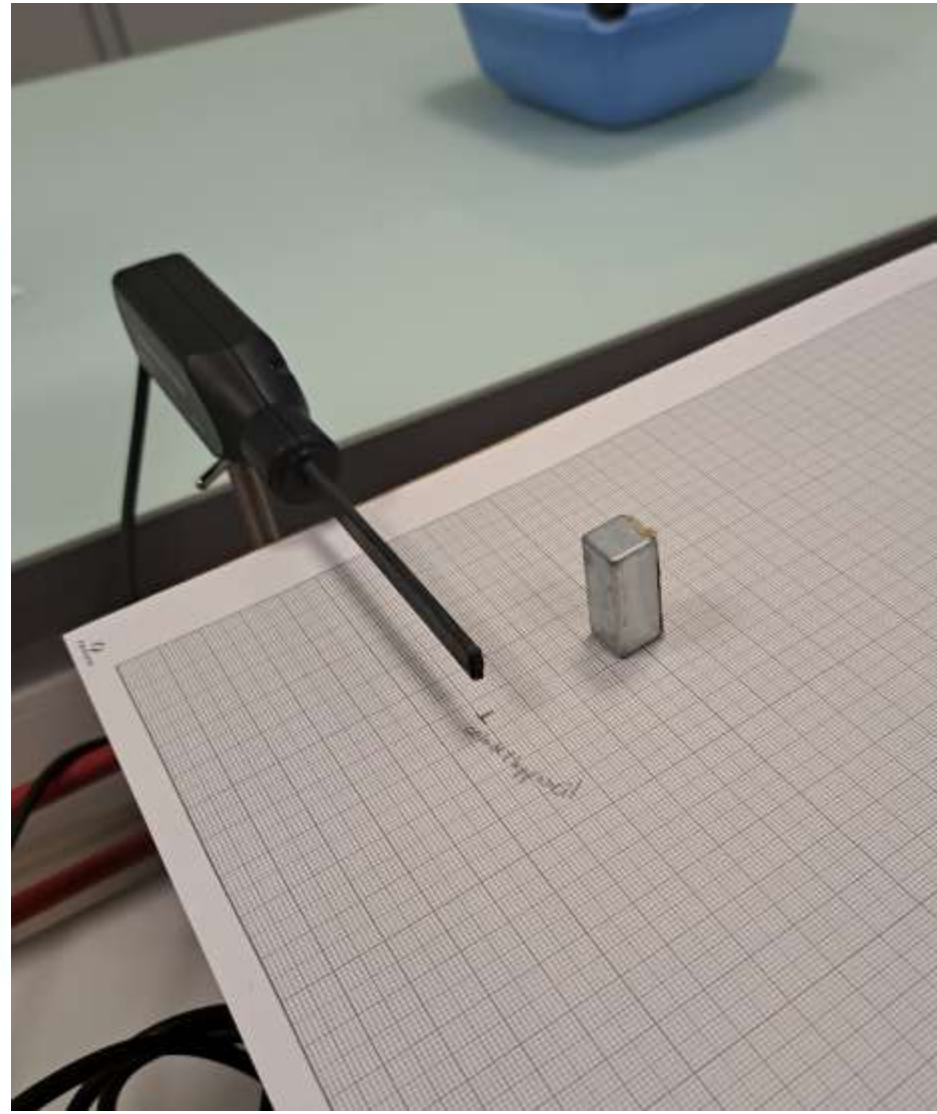
$$\frac{d\Phi}{dt} \propto \mathrm{S}\, \frac{d\mathrm{B}_n}{dt}$$

$donc$:

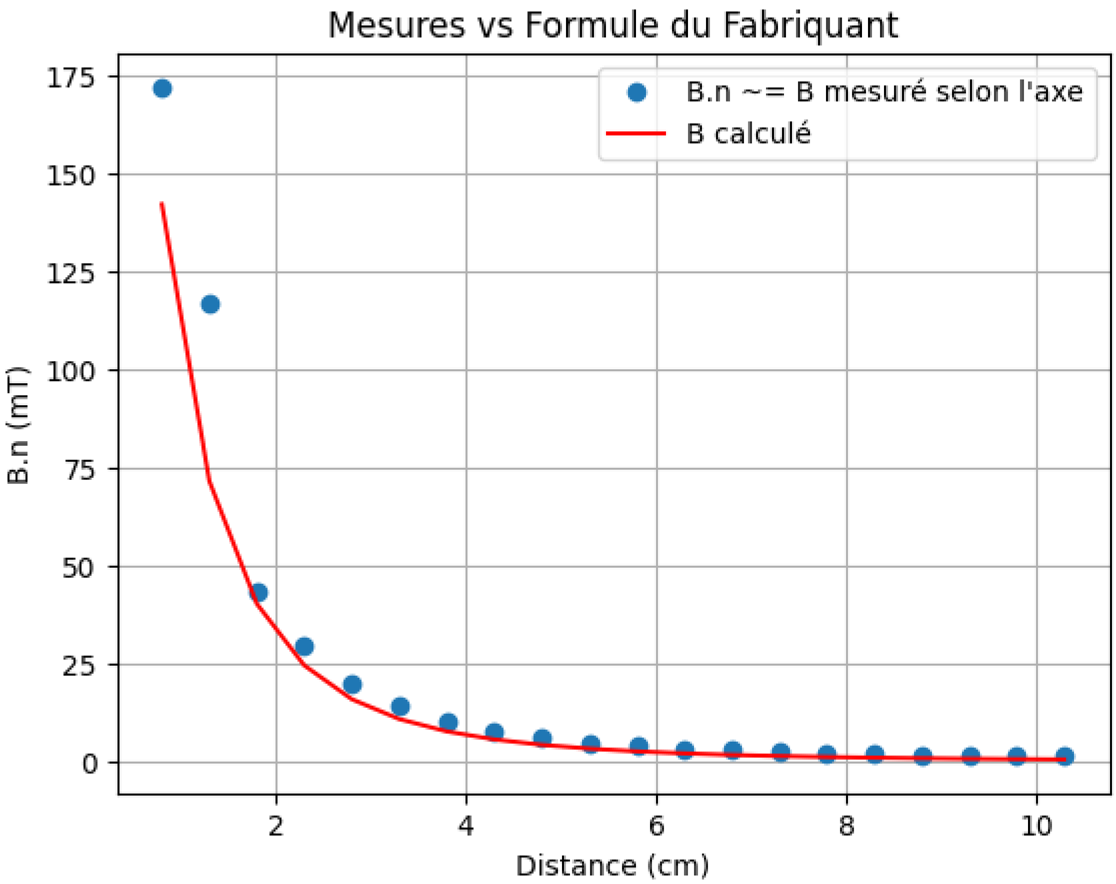
$$fem \propto \omega\, S\, B_0 \cos(\omega t)$$

EFFET DE LA DISTANCE

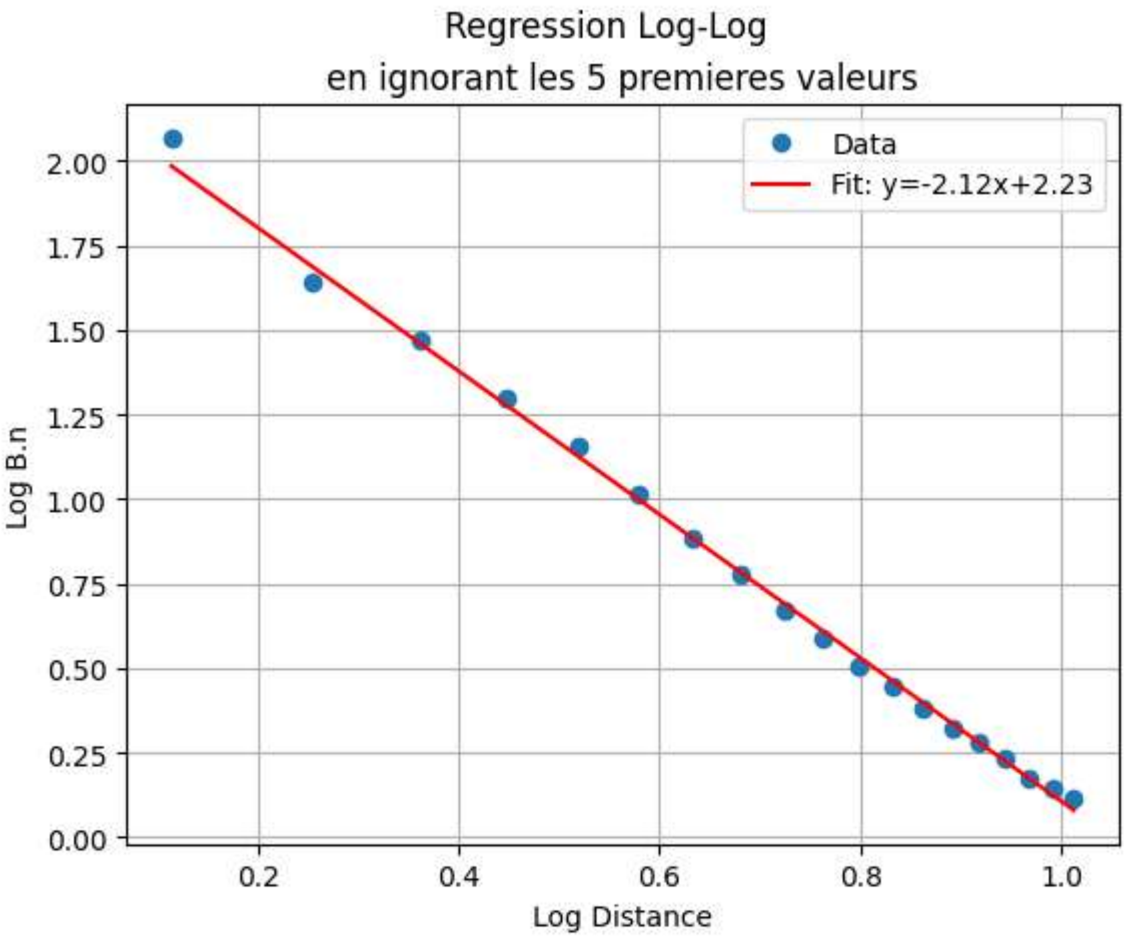




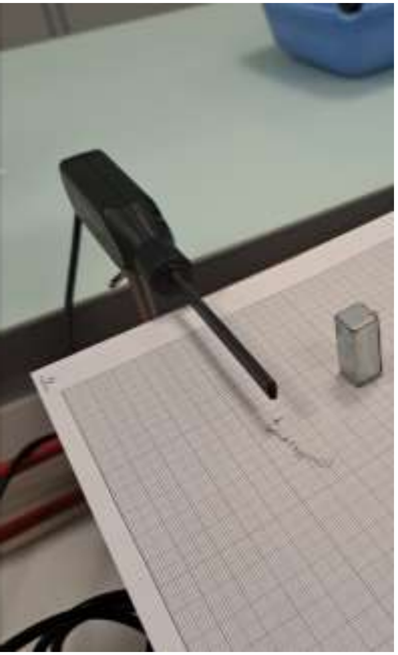
EFFET DE LA DISTANCE

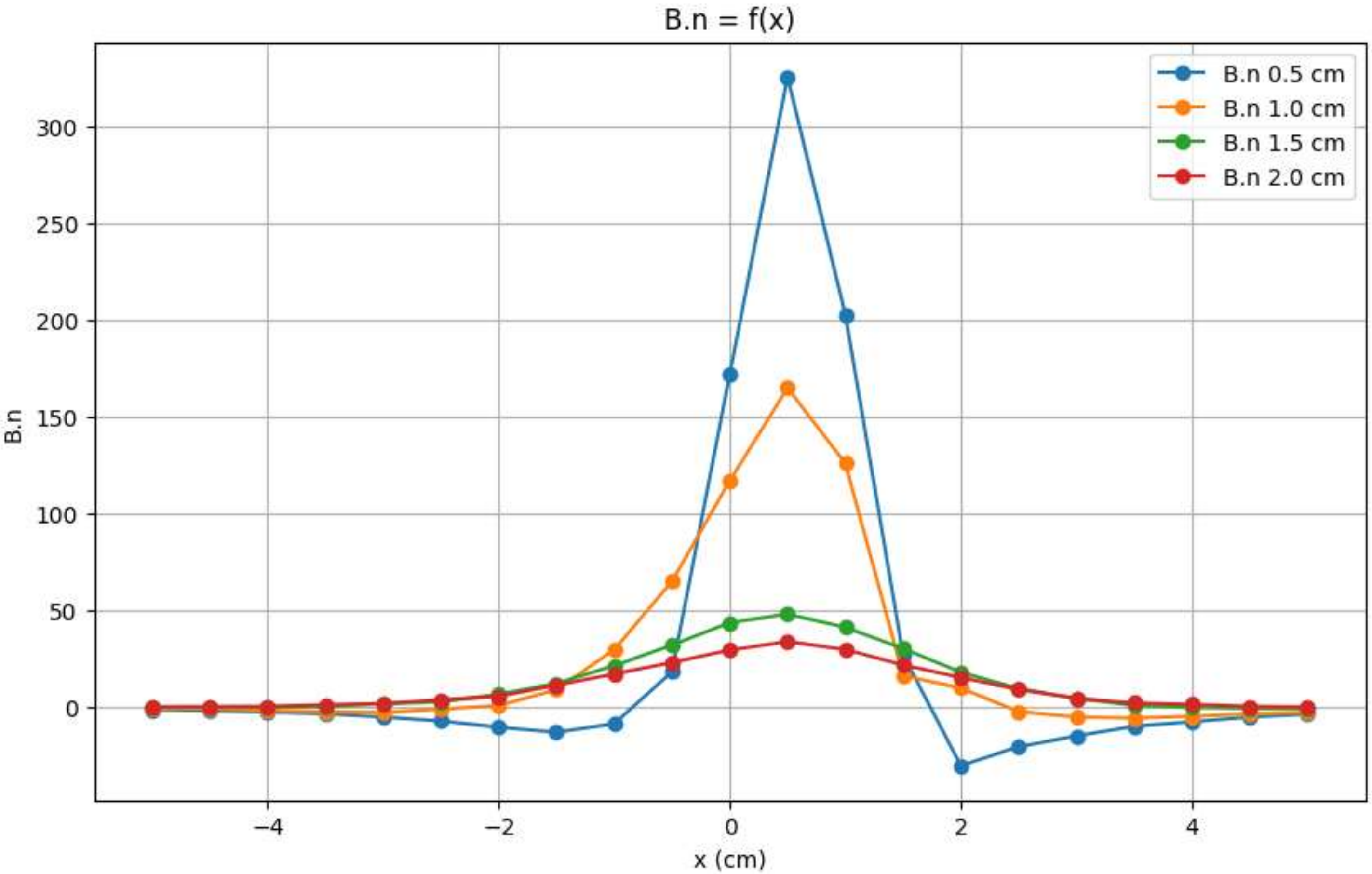


EFFET DE LA DISTANCE



SIMULATION SIMPLIFIÉE

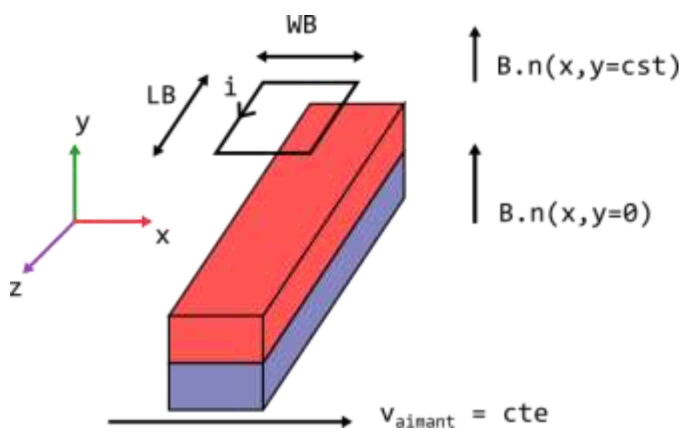




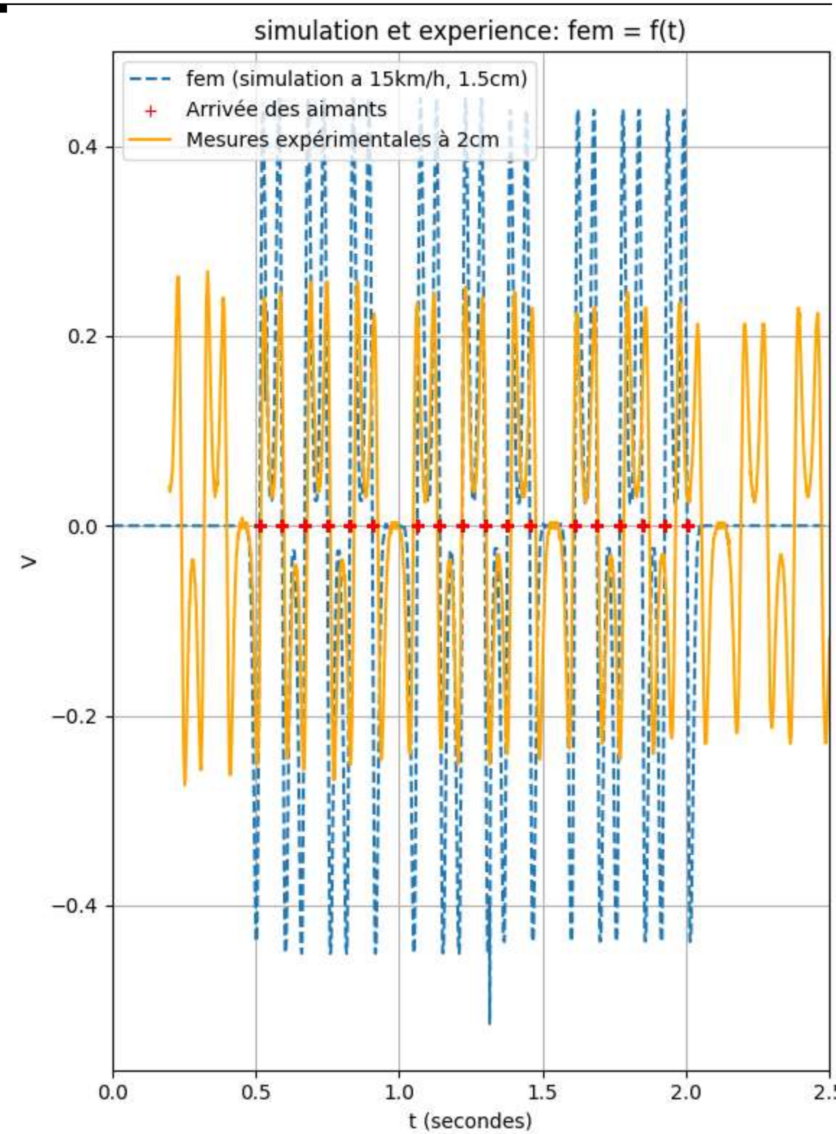
SIMULATION SIMPLIFIÉE

- Utilisation de données de mesure
- Prolongation sur les bords
- Intégration du mouvement et variation de flux avec quelques approximations.

La méthode d'Euler suffit.



$$\Phi(X) = \int_{-X-B_w/2}^{-X+B_w/2} B_n(y = cte, x)\, L_B dx$$



REALISATION DES OBJECTIFS

- Mettre en évidence et interpréter le phénomène d'induction entre une bobine et un aimant en mouvement.
- Déterminer et interpréter la vitesse minimale permettant le fonctionnement de l'éclairage.
- Réaliser une simulation numérique analogue à la dynamo réalisée
- Comparaison avec un modèle commercial.

ANNEXE FEMM:

- `mo_makeplot(PlotType,NumPoints,Filename,FileFormat)` Allows Lua access to the X-Y plot routines. If only `PlotType` or only `PlotType` and `NumPoints` are specified, the command is interpreted as a request to plot the requested plot type to the screen. If, in addition, the `Filename` parameter is specified, the plot is instead written to disk to the specified file name as an extended metafile. If the `FileFormat` parameter is also, the command is instead interpreted as a command to write the data to disk to the specfied file name, rather than display it to make a graphical plot. Valid entries for `PlotType` are:

`PlotType`	Definition
0	Potential
1	$\lvert B \rvert$
2	$B \cdot n$
3	$B \cdot t$

Valid file formats are

`FileFormat`	Definition
0	Multi-column text with legend
1	Multi-column text with no legend
2	Mathematica-style formatting

ANNEXE FEMM:

```python
#integrate thie radial field intensity over a surface S(r)
#X being the radius, Y being the normal intensity at that radius

def Phi(r): #r<0.14m = 14cm
    P = 0
    for i in range(len(X)-1):
        if X[i+1]-X[0]<=r:
            dr = X[i+1] - X[i]
            dr = dr*1e-2 #convertir en m
            localr = X[i+1]*1e-2 #convertir en m
            P += (Y[i] + Y[i+1])/2 *(2*np.pi)*localr*dr
        else:
            break
    return P
```

✓ 0.0s

```python
plt.figure()
plt.title("$\Phi _{S(r, d=1.2cm)}$")
plt.plot(X,[Phi(a)*1e3 for a in X])
plt.xlabel("r (cm)")
plt.ylabel("$\Phi$ (mWb)")
plt.show()
```

✓ 0.1s

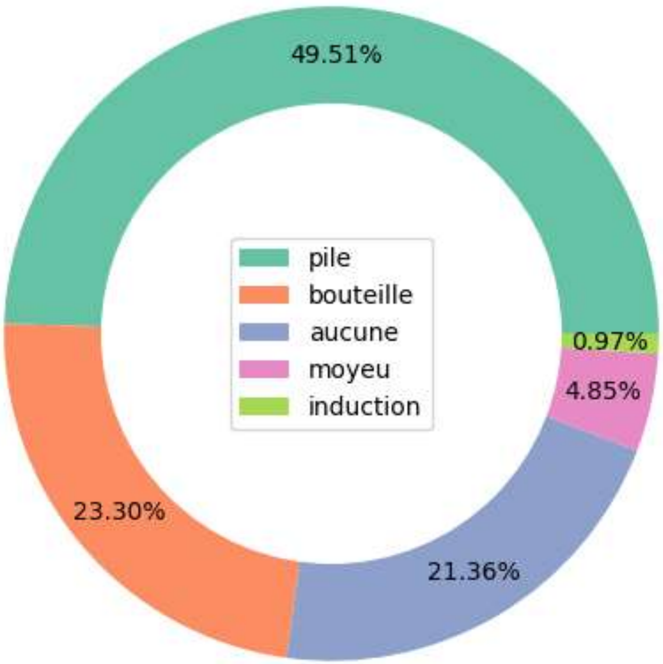
ANNEXE FEMM

```python
#integrate thie radial field intensity over a surface S(r)
#X being the radius, Y being the normal intensity at that radius

def Phi(r): #r<0.14m = 14cm
    P = 0
    for i in range(len(X)-1):
        if X[i+1]-X[0]<=r:
            dr = X[i+1] - X[i]
            dr = dr*1e-2 #convertir en m
            localr = X[i+1]*1e-2 #convertir en m
            P += (Y[i] + Y[i+1])/2 *(2*np.pi)*localr*dr
        else:
            break
    return P
```

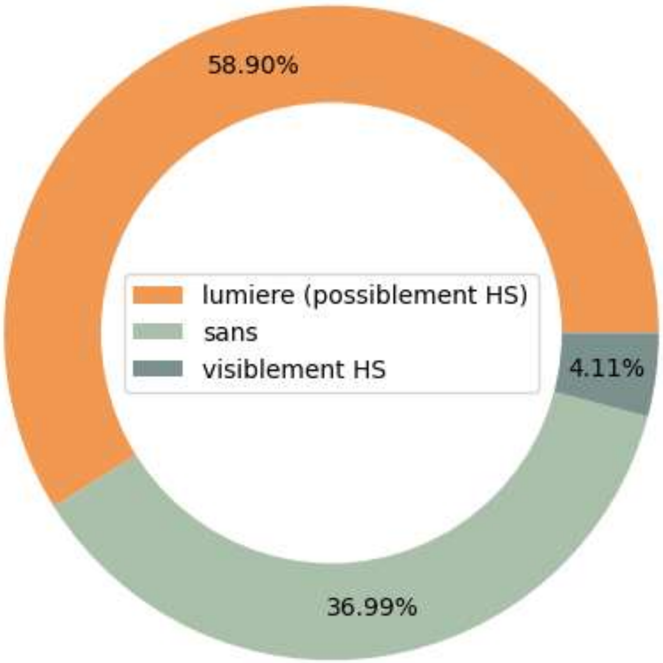
```python
plt.figure()
plt.title("$\Phi _{S(r, d=1.2cm)}$")
plt.plot(X,[Phi(a)*1e3 for a in X])
plt.xlabel("r (cm)")
plt.ylabel("$\Phi$ (mWb)")
plt.show()
```

ANNEXE: Recensement – *Gare de Versailles Chantiers*

Février 2025 ~ 100 vélos présents



Dispositifs Utilisés



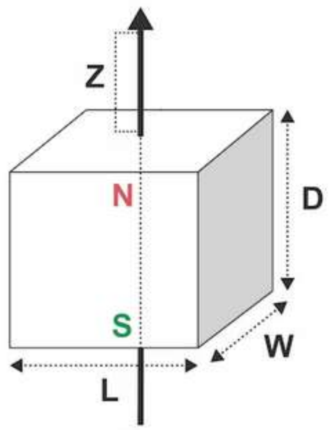
Lumières sur échantillon

FORMULE DISTANCE

Formule pour la densité de flux parallélépipède magnétique

Formule pour calculer le champ B sur l'axe de symétrie d'un parallélépipède ou d'un cube magnétique axialement magnétisé :

$$B = \frac{B_r}{\pi}\left[arctan\left(\frac{LW}{2z\sqrt{4z^2+L^2+W^2}}\right) - arctan\left(\frac{LW}{2(D+z)\sqrt{4(D+z)^2+L^2+W^2}}\right)\right]$$



$\boldsymbol{B_r}$: Champ rémanent, indépendant de la géométrie de l'aimant (voir Données physiques de l'aimant)

z : Distance de la surface du pôle sur l'axe de symétrie

L : Longueur du parallélépipède

W : Largeur du parallélépipède

D : Épaisseur (ou hauteur) du parallélépipède

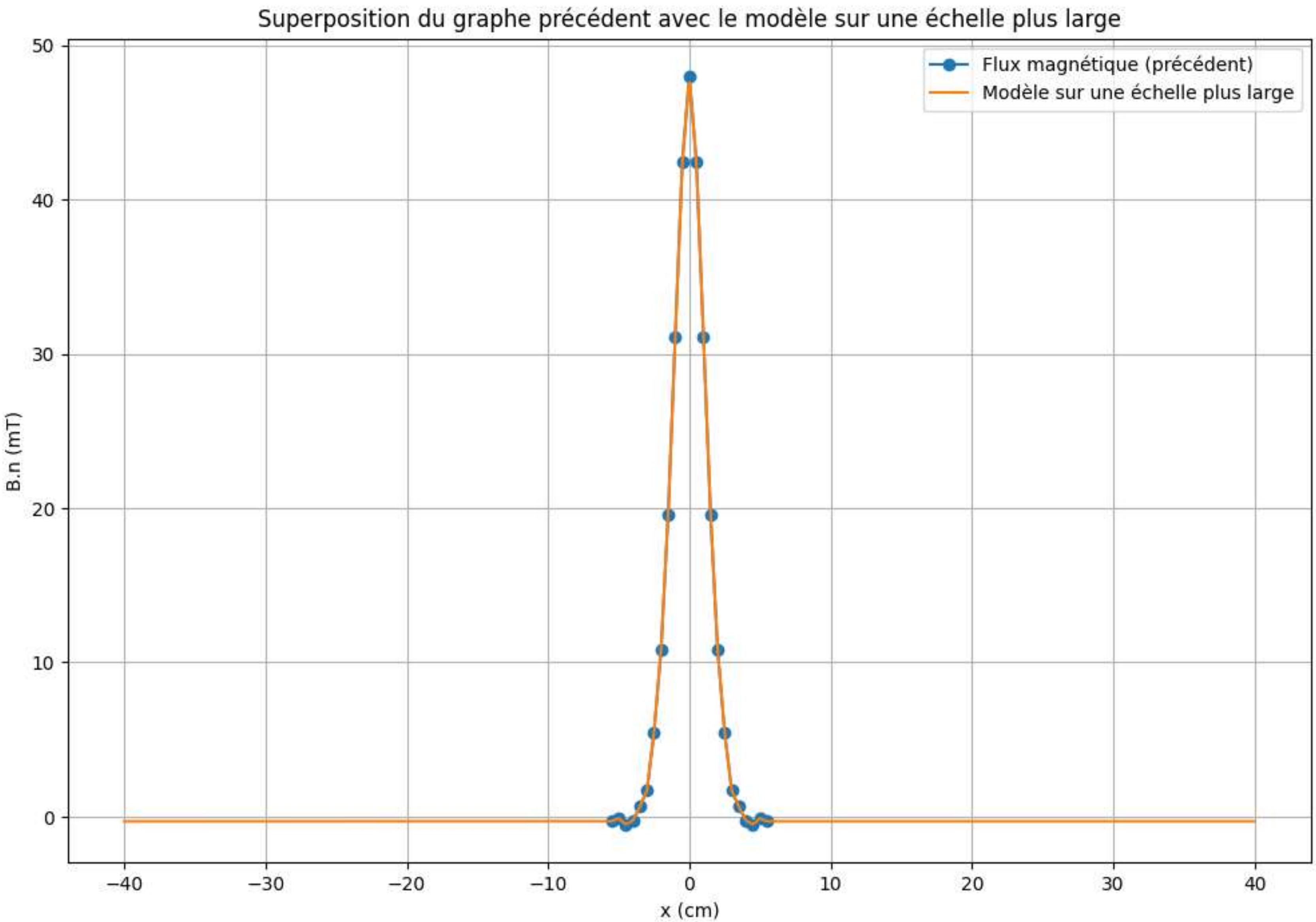
L'unité de longueur peut être choisie librement, mais à condition qu'elle soit la même pour toutes les longueurs.

https://www.supermagnete.fr/faq/Comment-calculer-la-densite-du-flux-magnetique

ANNEXES

```python
from scipy.interpolate import interp1d

def get_field_at_distance(offsetx, dist_array, vals_array):
    # Vérifie si l'offset est dans les limites
    # Si on est en dehors, prendre la valeur constante la plus proche
    # c'est légitime du momt qu'un autre aimant vient apres et que son champ prédomine devant les valeurs réelles attendues, d'ordre inférieur à cette constante.
    if offsetx < dist_array[0]:
        return vals_array[0]  # Valeur du bord gauche
    elif offsetx > dist_array[-1]:
        return vals_array[-1]  # Valeur du bord droit
    else:
        # Interpolation linéaire dans les bornes de mes mesures
        interpolator = interp1d(dist_array, vals_array, kind='linear', fill_value="extrapolate")
        return interpolator(offsetx)
```

Prolongement du champ



Superposition du graphe précédent avec le modèle sur une échelle plus large
Flux magnétique (précédent)
Modèle sur une échelle plus large
B.n (mT)
x (cm)
50
40
30
20
10
0
−40
−30
−20
−10
0
10
20
30
40

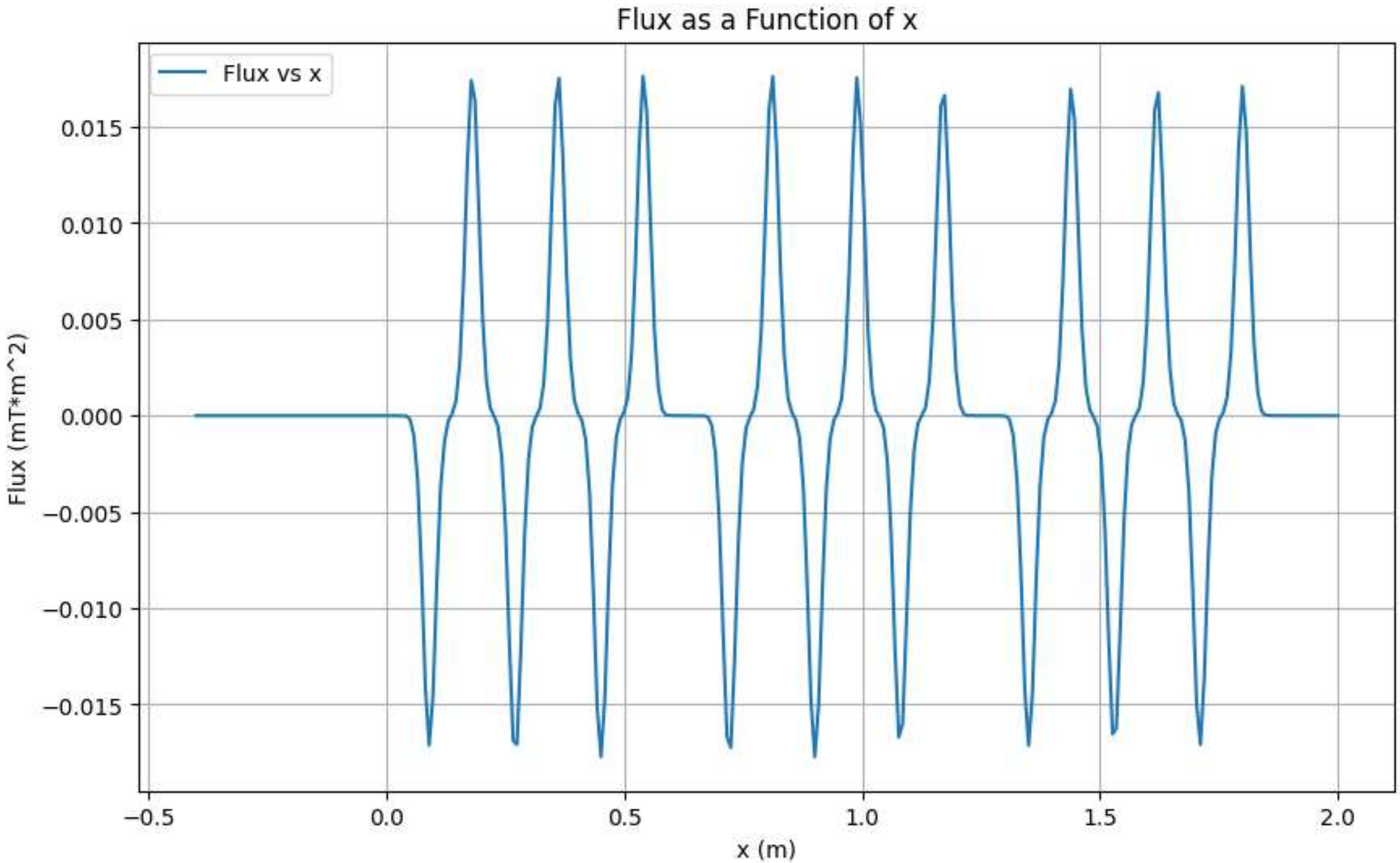
```
#dimensionnement de la bobine
#2x2cm selon les axes respectifs z et x
BW=2e-2
BL=2e-2

def get_flux(mag_funct, offsetx, deltax=1e-3):#1mm dej bien, 0.1 un peu extreme
    # deltax est la taille de la cellule
    # mag_funct est une fonction qui prend en entrée la distance et renvoie la valeur du champ magnétique
    # on suppose B constant // à z, fonction de x
    # on intègre sur x et z
    flux = 0
    x_values = np.arange(-BW/2-offsetx, BW/2 - offsetx + deltax, deltax) #valeurs de x sur lesquelles on integre
    for x in x_values:
        # On suppose que la fonction mag_funct est définie pour renvoyer le champ magnétique à une distance donnée
        B = mag_funct(x)  # On suppose que mag_funct est une fonction de x
        flux += B * deltax * BL  # Contribution de chaque cellule

    return flux

N_magnets = 21 # Nombre total d'aimants
group_mag = 7 #skip every 7th magnet
mag_offset = 9e-2 #décalage entre les aimants en m
mag_positions = []
def field_func(x):
    #somme les contributions de chaque aimant
    total_field = 0
    for i in range(N_magnets):
        # Position de l'aimant i
        if i % group_mag == 0:
            continue
        j=i-i//group_mag  # Skip every 7th magnet

        magnet_position = -i * mag_offset
        mag_positions.append(magnet_position)

        # Contribution de l'aimant i
        total_field += ((-1)**j)*get_field_at_distance(x - magnet_position, dist_trunc*1e-2, vals_trunc)
    return total_field
```

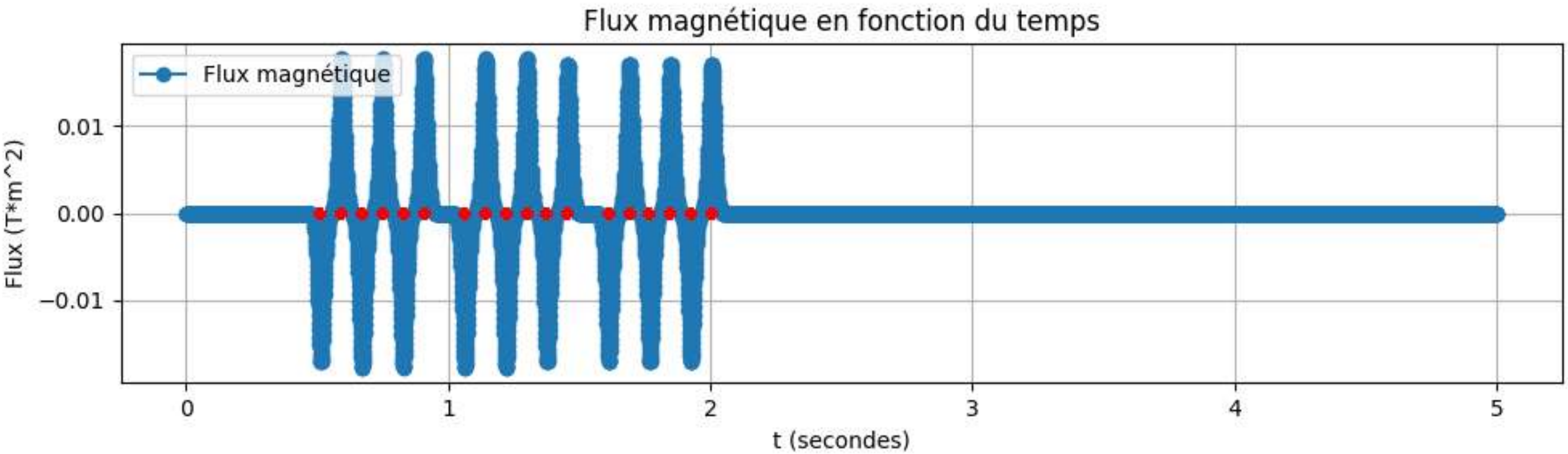
Superposition des champs et calcul de flux

```python
# Simulation de l'effet du mouvement de l'aimant
# Parametres de la simulation ! attention parametres et ocnfig des aimants au dessus
RAYON_ROUE = 0.69/2 #m
RAYON_AIMANT = 95e-3 #mm soit 9.5cm
vitesse_velo = 15 #kmph
vitesse_velo = vitesse_velo * 1000 / 3600  # Convertir en m/s
omega = vitesse_velo / RAYON_ROUE  # Vitesse angulaire en rad/s
v = omega * RAYON_AIMANT  # Vitesse linéaire de l'aimant en m/s
# Initialisation de la simulation
Tmax = 5  # 5 secondes de sim
N=5000 #nb de pas de temps
T = np.linspace(0, Tmax, N)  # Valeurs de deltax à tester
dt = Tmax/(N-1)
print(f"simulation pour v_aimant = {v:.2e} m/s, omega = {omega:.2f} rad/s, soit une vitesse au sol de {vitesse_velo:.2f} m/s")
# v = 0.5  # Vitesse de déplacement en m/s
startx = -0.5 #position initiale, arrivée du 1er aimant apres 1seconde
x = startx  # Position initiale
flux_values = []
for t in tqdm(T):
    flux = get_flux(field_func, x)
    flux_values.append(flux)
    deltax = v * dt
    x += deltax  # Mise à jour de la position
```

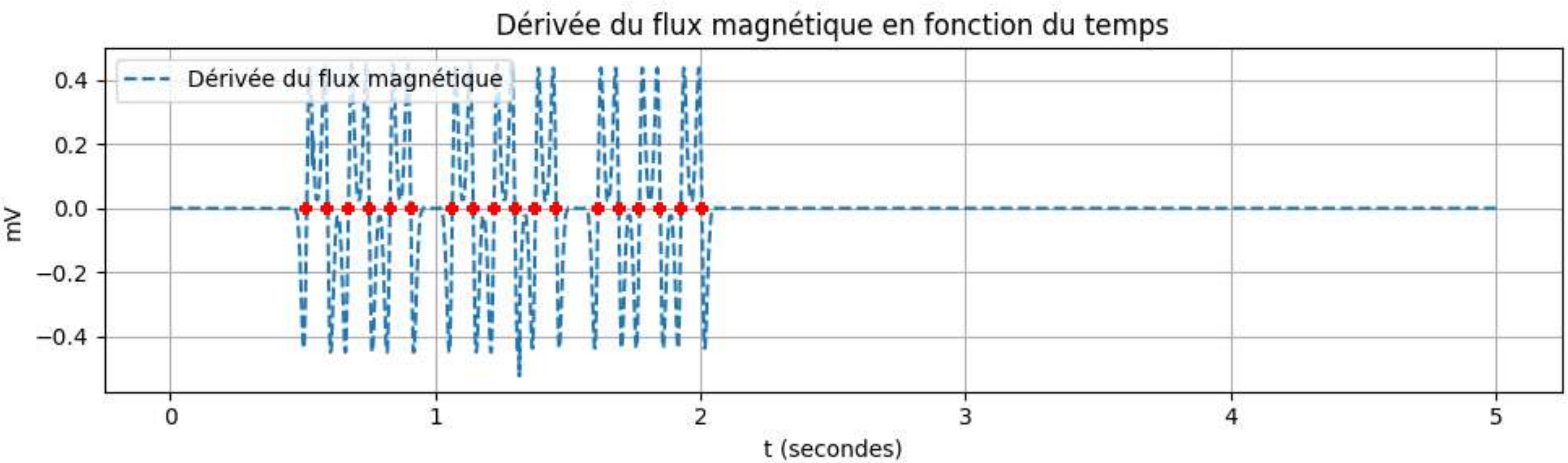
flux = f(t) pour une vitesse de passage donnée

```
simulation pour v_aimant = 1.15e+00 m/s, omega = 12.08 rad/s, soit une vitesse au sol de 4.17 m/s
100%|██████████| 5000/5000 [00:09<00:00, 536.58it/s]
```


Flux as a Function of x
Flux vs x
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
Flux (mT*m^2)
−0.5
0.0
0.5
1.0
1.5
2.0
x (m)

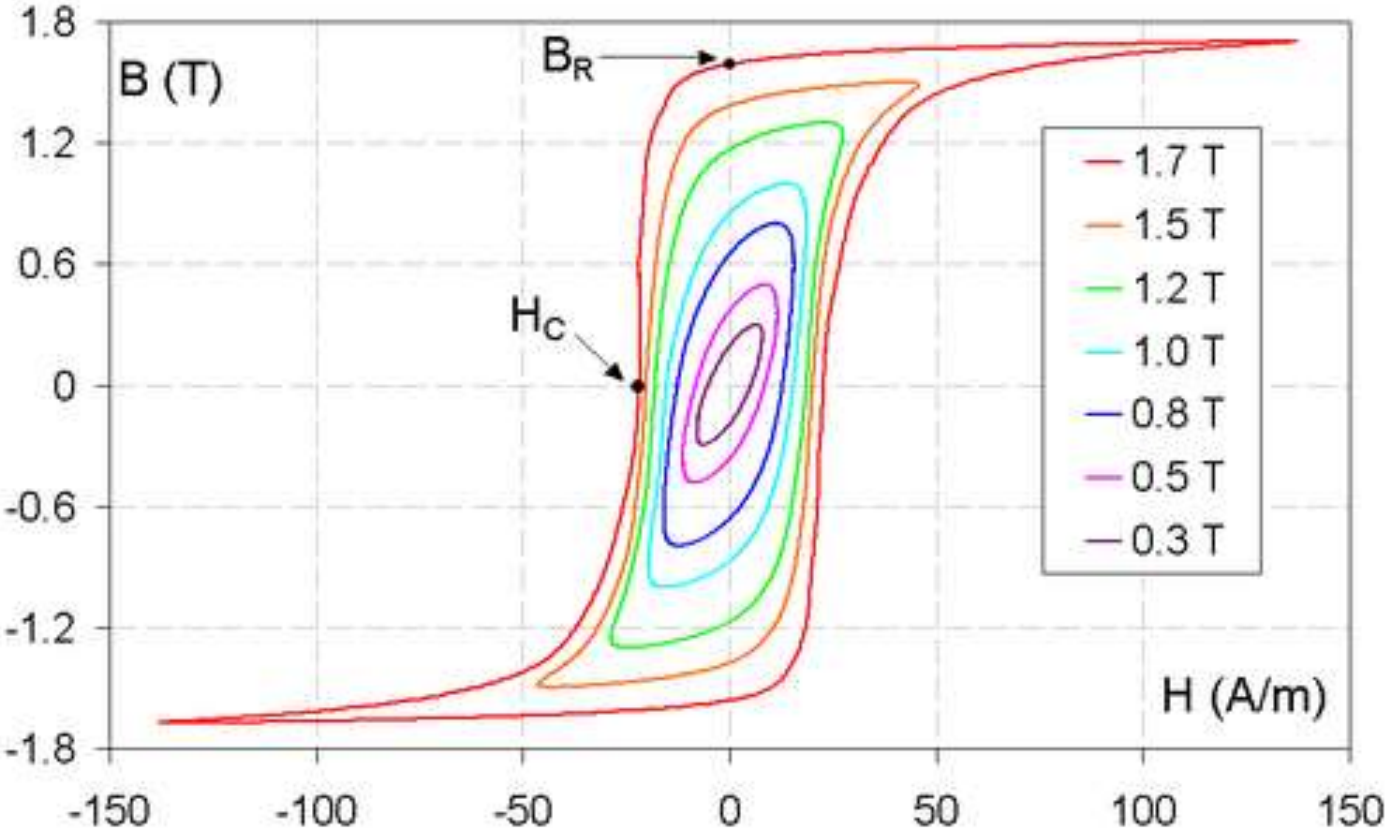


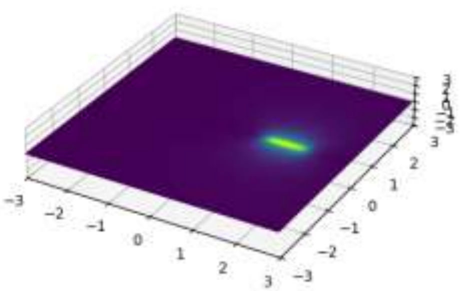
Flux magnétique en fonction du temps
Flux magnétique
0.01
0.00
−0.01
Flux (T*m^2)
0
1
2
3
4
5
t (secondes)

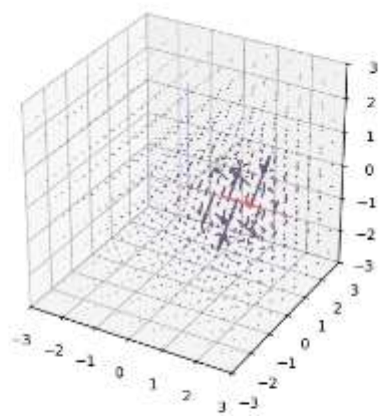


Dérivée du flux magnétique en fonction du temps
Dérivée du flux magnétique
0.4
0.2
0.0
−0.2
−0.4
mV
0
1
2
3
4
5
t (secondes)

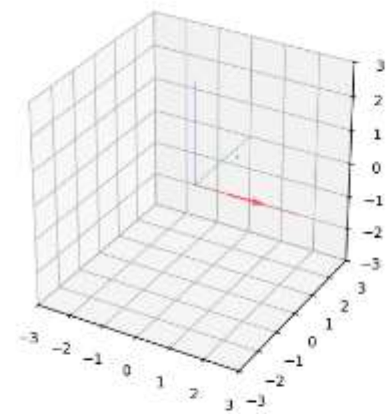
```python
# Load the measurement data from the CSV file
measurement_data = np.loadtxt('../data/mesure2cm_bis_3.csv', delimiter=',', skiprows=1)
N_tours = 350 # Nombre de tours de bobine
# Extract time and voltage values
time_measurement = measurement_data[:, 0]
voltage_measurement = measurement_data[:, 1]
fem_calc = flux_derivative * N_tours

# Plot the derivative and superimpose the measurement data
plt.figure(figsize=(6, 8))
plt.plot(T, flux_derivative, marker='', linestyle='--', label='fem (simulation a 15km/h, 1.5cm)')
plt.scatter(magnet_times, [0]*len(magnet_times), color="red", marker="+", zorder=10, label='Arrivée des aimants', linewidths=0.8)
plt.plot(time_measurement+0.20, voltage_measurement, marker='', linestyle='-', label='Mesures expérimentales à 2cm', color='orange')
plt.xlim(0,2.5)
plt.xlabel('t (secondes)')
plt.ylabel('U (V)')
plt.title('simulation et experience: fem = f(t)')
plt.legend(loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```


B (T)
H (A/m)
B_R
H_C
1.8
1.2
0.6
0
-0.6
-1.2
-1.8
-150
-100
-50
0
50
100
150
1.7 T
1.5 T
1.2 T
1.0 T
0.8 T
0.5 T
0.3 T





Ici pour un fil de courant en vérifiant les résultats par théorème de superposition par exemple.
Mais…



Superposition d'un fil de courant I et –I : champ nul.

ANNEXES

```python
def int_val(A,B,C):
    return (C-B)/np.sqrt(A-2*B+C)+B/np.sqrt(A)

def mag(ra,rb,r2):
    A = (r2-ra)
    A = np.dot(A,A)
    B = (r2-ra)
    B = np.dot(B,rb-ra)
    C = (rb-ra)
    C = np.dot(C,C)
    k = mu0*I/(4*pi)
    v = np.cross(rb-ra, r2-ra)
    return k*int_val(A,B,C)*v/(np.dot(A,C)-np.dot(B,B)+1e-10)
```

Implémentation fil de courant