



MASTER 2 INGÉNIERIE MATHÉMATIQUE - OPTION INUM

---

## Scientific Machine Learning :

Résolution des Équations de Maxwell en 1D avec PINNs  
et DeepXDE

---

**Auteurs :**  
Yannick KOULONI et Qinyan YANG

18 mars 2025

## Table des matières

<b>1 Problème 1 :Mode stationnaire dans une cavité . . . . .</b>	<b>2</b>
1.1 Contexte . . . . .	2
1.2 Résultats et Analyses dans le cas où le mode $m = 1$ . . . . .	2
1.2.1 Comparaison de la solution approchée avec la solution analytique . . . . .	2
1.2.2 Analyse comparative des résultats avec différents paramètres d'échantillonnage des données d'entraînement . . . . .	3
1.2.3 Analyse comparative des résultats selon le nombre d'époques	5
1.2.4 Analyse comparative des résultats avec différents réseaux de neurones . . . . .	7
1.3 Résultats et Analyses dans le cas où le mode $m = 2$ . . . . .	10
1.3.1 Comparaison de la solution approchée avec la solution analytique . . . . .	10
1.3.2 Résultats optimal . . . . .	10
<b>2 Problème 2 :Propagation d'impulsion dans un domaine périodique homogène . . . . .</b>	<b>11</b>
2.1 Contexte . . . . .	11
2.2 Solutions numériques obtenue par PINNs . . . . .	13
<b>3 Annexes : Codes pour la mise en oeuvre . . . . .</b>	<b>15</b>
3.1 Annexe 1 : mise en oeuvre du problème 1 . . . . .	15
3.2 Annexe 3 : mise en oeuvre du problème 2 . . . . .	37

# Introduction

Les équations de Maxwell décrivent le comportement des champs électromagnétiques. Ce projet vise à résoudre les équations de Maxwell en 1D à l'aide des réseaux de neurones informés par la physique (PINNs) et de la bibliothèque DeepXDE. Nous utilisons une approche basée sur l'apprentissage automatique scientifique pour approximer la solution du problème de mode stationnaire dans une cavité 1D fermée avec des parois métalliques parfaites et ensuite pour approximer celle du problème de propagation d'impulsion dans un domaine périodique homogène.

## 1 Problème 1 :Mode stationnaire dans une cavité

### 1.1 Contexte

Nous considérons les équations de Maxwell en 1D dans le domaine temporel :

$$\varepsilon_r \frac{\partial E(x, t)}{\partial t} - \frac{\partial H(x, t)}{\partial x} = 0, \quad (1)$$

$$\mu_r \frac{\partial H(x, t)}{\partial t} - \frac{\partial E(x, t)}{\partial x} = 0. \quad (2)$$

On définit  $\Omega = [0, 1]$ .

Dans la suite du travail, on prendra  $\epsilon_r(x) = \mu_r(x) = 1, \forall x \in \Omega$

Les conditions aux limites sont définies par :

$$E(x = 0, t) = E(x = 1, t) = 0, \quad \forall t \in [0, 2T]$$

La solution analytique pour le mode stationnaire dans la cavité est donnée par :

$$E_a(x, t) = \sin(\omega x) \cos(\omega t), \quad (3)$$

$$H_a(x, t) = \cos(\omega x) \sin(\omega t), \quad (4)$$

où  $\omega = m\pi$  est la pulsation du mode et la période  $T = \frac{2\pi}{\omega}$

### 1.2 Résultats et Analyses dans le cas où le mode $m = 1$

#### 1.2.1 Comparaison de la solution approchée avec la solution analytique

Dans le cas  $m = 1$ , où le domaine temporel est  $[0, 4]$ , après entraînement du modèle PINNs, nous obtenons la figure 1 pour visualiser la solution obtenue par PINNs et la solution analytique.

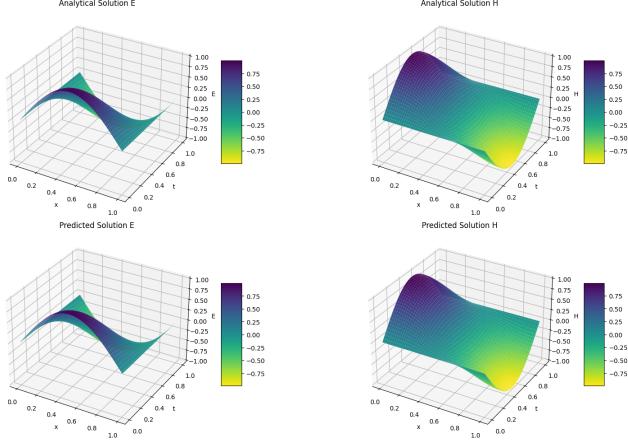


FIGURE 1 – La solution PINNs et la solution analytique.

Nous observons que la solution obtenue par le modèle PINNs est presque en accord avec la solution analytique. Cependant, des améliorations peuvent être apportées en augmentant le nombre de neurones ou en ajustant les paramètres d’entraînement. Pour mieux comprendre l’influence des paramètres, dans la suite on se concentre à observer la variation de qu’un seul point au cours du temps.

### 1.2.2 Analyse comparative des résultats avec différents paramètres d’échantillonnage des données d’entraînement

Dans cette section, nous étudions l’influence des paramètres d’échantillonnage des données d’entraînement sur les résultats. Nous comparons les performances et les erreurs pour trois configurations distinctes :

- Configuration 1 : `num_domain = 300, num_boundary = 30, num_initial = 30`
- Configuration 2 : `num_domain = 600, num_boundary = 60, num_initial = 60`
- Configuration 3 : `num_domain = 1000, num_boundary = 100, num_initial = 100`

En utilisant un nombre d’époques fixé à 6000, ainsi qu’un réseau de neurones composé de 4 couches et de 50 neurones par couche, nous obtenons les résultats suivants pour chaque configuration :

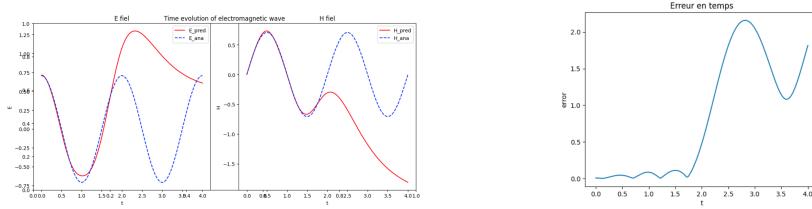


FIGURE 2 – Configuration 1 : dom=300, bd=30, init=30

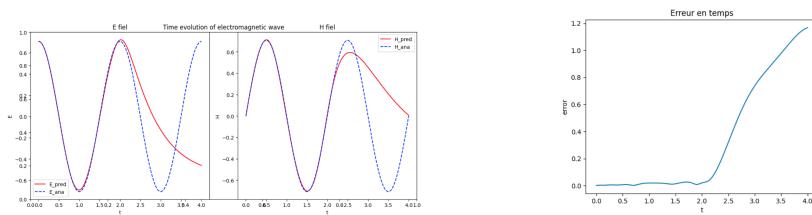


FIGURE 3 – Configuration 2 : dom=600, bd=60, init=60

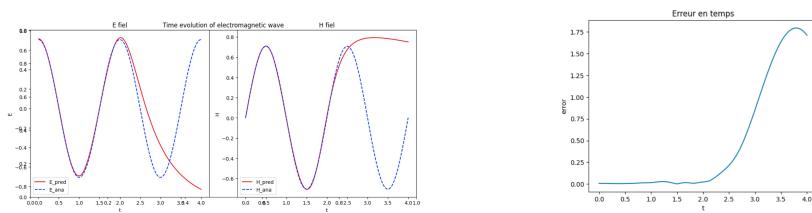
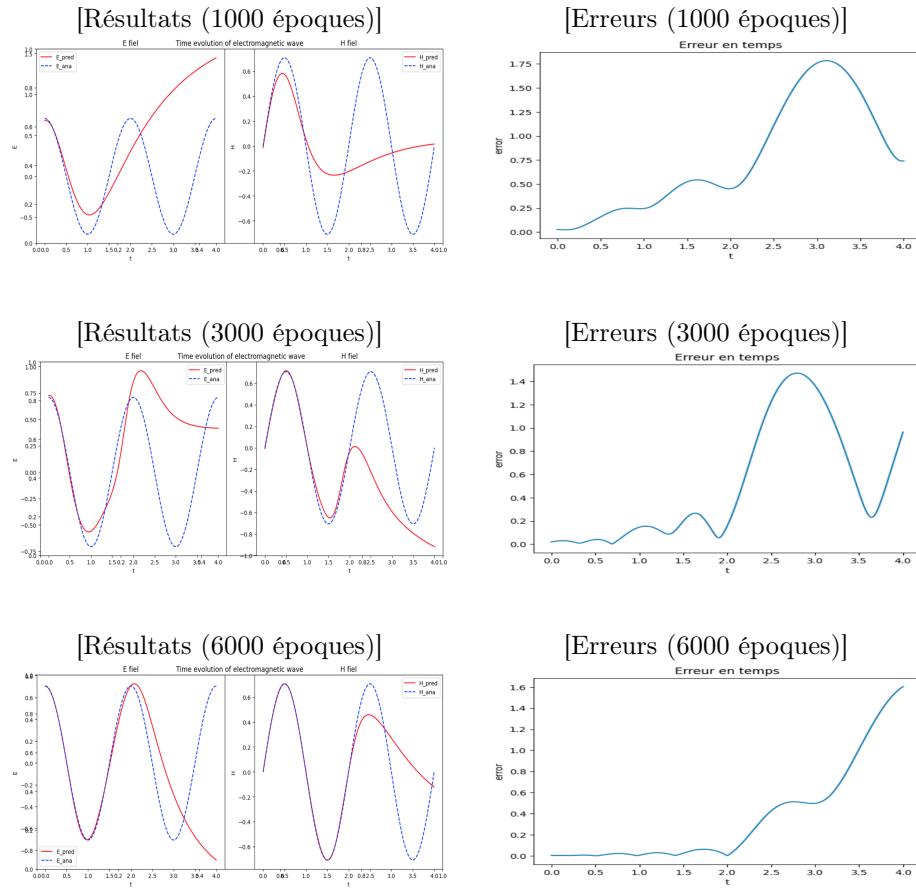


FIGURE 4 – Configuration 3 : dom=1000, bd=100, init=100

L'étude paramétrique des configurations démontre qu'à partir de la Configuration 2, le modèle présente une convergence optimale dans la zone d'entraînement, avec une marge d'erreur acceptable. L'analyse démontre un compromis clair : l'augmentation des échantillons frontaliers (`num_boundary`) améliore la stabilité des prédictions, tandis que les échantillons de domaine (`num_domain`) influencent directement la précision globale.

### 1.2.3 Analyse comparative des résultats selon le nombre d'époques

Dans cette section, nous étudierons l'influence des résultats en fonction du nombre d'époques. Nous analyserons les performances et les erreurs obtenues avec 1000, 3000, 6000 et 12000 époques.



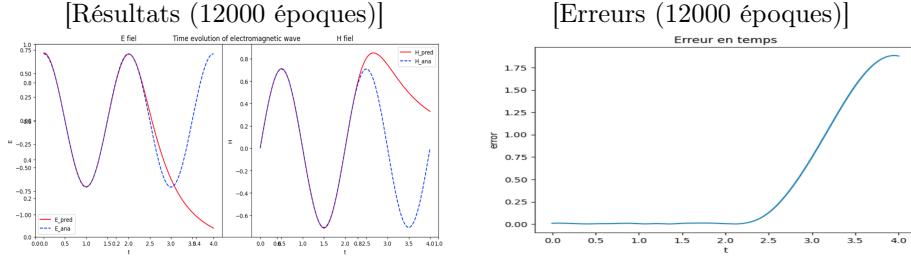


FIGURE 5 – Comparaison des résultats et erreurs pour différentes époques

L'erreur maximale dans le domaine d'entraînement aux époques 1000, 3000, 6000 et 12000 est respectivement de 0.54322085, 0.26697367, 0.06265177 et 0.01245142. Cette étude met en évidence une amélioration exponentielle des performances avec l'augmentation du nombre d'époques, suivie d'un gain marginal décroissant au-delà de 6 000 époques. Un plateau de précision est atteint dès 10 000 époques, suggérant qu'un compromis optimal entre coût et bénéfice se situe autour de 6 000 époques pour des applications pratiques. Par conséquent, nous retenons 6 000 époques pour la suite de notre étude.

Des tableaux récapitulatifs ci-après :

Époque	Erreur maximale
1000	0.54322085
3000	0.26697367
6000	0.06265177
12000	0.01245142

TABLE 1 – Erreur maximale aux différentes époques

Étape	Train Loss	Test Loss	Test Metric	Temps d'entraînement (s)
1000	1.54e-02	1.56e-02	[3.66e-01]	13.681160
3000	9.34e-04	1.06e-03	[5.68e-02]	39.962715
6000	6.53e-05	7.17e-05	[1.10e-02]	80.757528
12000	1.79e-05	2.11e-05	[6.93e-03]	154.333011

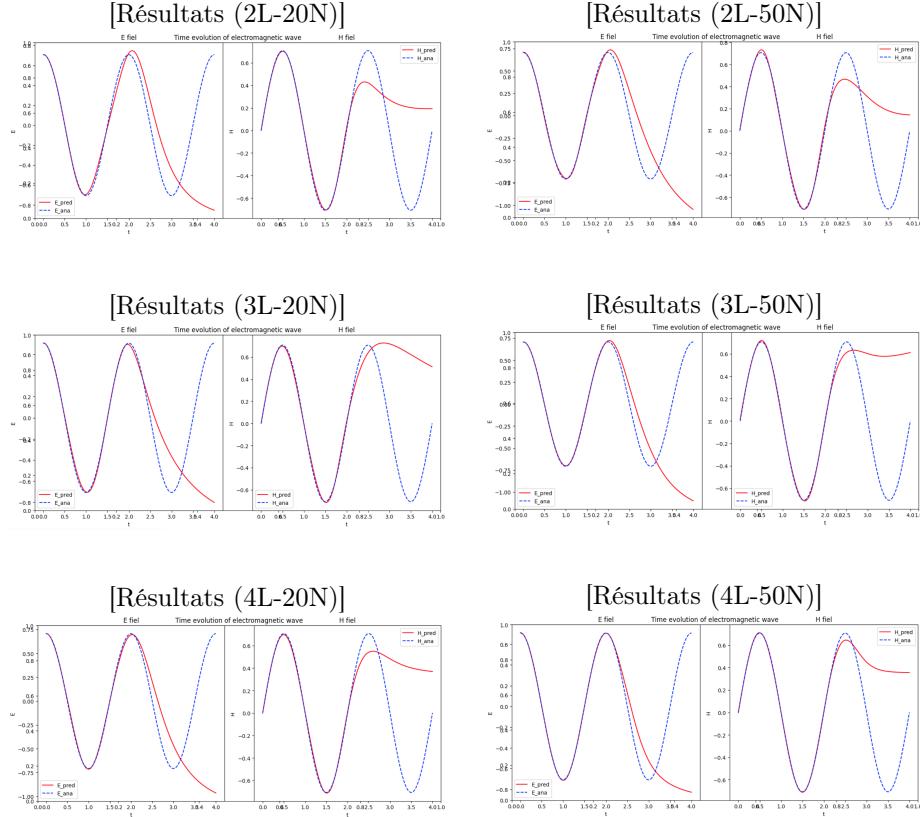
TABLE 2 – Évolution des performances du modèle au fil des époques

### 1.2.4 Analyse comparative des résultats avec différents réseaux de neurones

Dans cette section, nous étudions l'impact des réseaux de neurones sur les résultats. Nous comparons les performances et les erreurs en fixant le nombre d'époques à 6000, ainsi que les paramètres d'échantillonnage des données d'entraînement suivants : `num_domain=1000`, `num_boundary=100` et `num_initial=100`. Les cas suivants sont analysés :

- Cas 1 : 2 couches et 20 neurones
- Cas 3 : 3 couches et 20 neurones
- Cas 5 : 4 couches et 20 neurones
- Cas 7 : 4 couches et 80 neurones
- Cas 2 : 2 couches et 50 neurones
- Cas 4 : 3 couches et 50 neurones
- Cas 6 : 4 couches et 50 neurones

Nous avons les résultats suivants pour chaque cas :



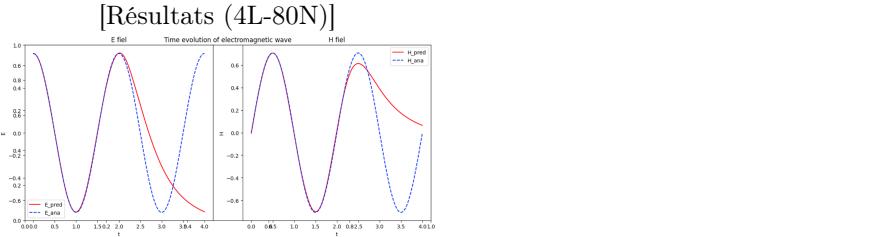


FIGURE 6 – Comparaison des résultats pour différents réseaux de neurones

Tracé de l'erreur en fonction du temps pour chaque cas :

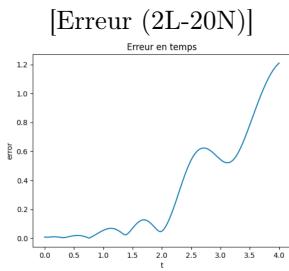


FIGURE 7 – erreur maximale = 0.12667

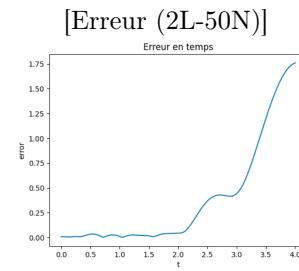


FIGURE 10 – erreur maximale = 0.04297

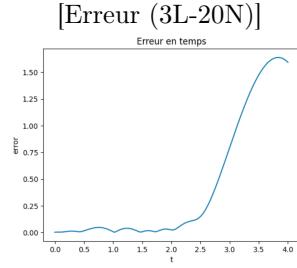


FIGURE 8 – erreur maximale = 0.04622

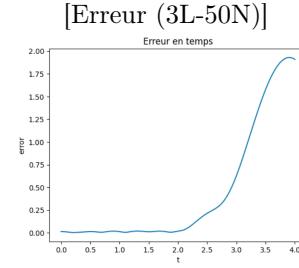


FIGURE 11 – erreur maximale = 0.01954

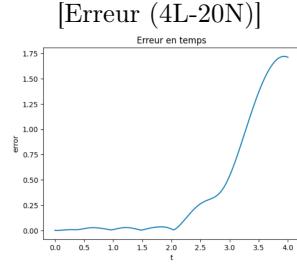


FIGURE 9 – erreur maximale = 0.03583

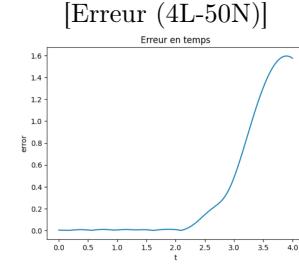


FIGURE 12 – erreur maximale = 0.01099

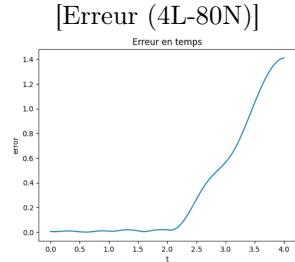


FIGURE 13 – erreur maximale = 0.02083

À travers l’analyse des résultats et des erreurs issus de diverses structures de réseaux de neurones, on constate que l’augmentation du nombre de couches de neurones et de neurones par couche améliore progressivement la précision de prédiction de l’évolution temporelle de l’onde électromagnétique, avec une réduction concomitante des erreurs. Ceci permet donc une meilleure approximation des données et une prédiction plus précise. Toutefois, il convient de souligner que l’augmentation de la complexité du modèle peut entraîner une augmentation du temps d’entraînement et un risque de surapprentissage. Par exemple, dans le cas d’un réseau à 4 couches et 80 neurones, l’erreur maximale est de 0,02083, légèrement supérieure à celle du cas à 4 couches et 50 neurones, mais reste à un niveau relativement faible. Cela indique que l’augmentation du nombre de neurones peut améliorer les performances du modèle, mais cela entraîne une hausse de la complexité de calcul et des coûts de calcul.

Dans le cas  $m = 1$ , nous pouvons utiliser `num_domain = 800, num_boundary = 85, num_initial = 85`, 6000 époques et un réseau de neurones de 4 couches et 50 neurones par couche pour optimiser notre modèle.

### 1.3 Résultats et Analyses dans le cas où le mode $m = 2$

#### 1.3.1 Comparaison de la solution approchée avec la solution analytique

Pour  $m = 2$  où le domaine temporel est  $[0, 2]$ , nous obtenons les résultats suivants :

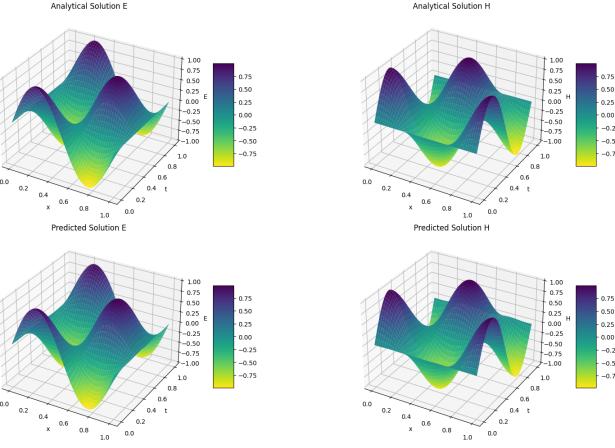


FIGURE 14 – La solution PINNs et la solution analytique.

Nous observons que la solution obtenue par le modèle PINNs est comme dans le cas  $m = 1$  presque en accord avec la solution analytique.

#### 1.3.2 Résultats optimal

Tout comme dans la section précédente, après avoir comparé différents paramètres d'entraînement, des réseaux de neurones et des époques, nous avons constaté que lorsque  $m=2$ , avec 15 000 époques, un réseau de neurones de 5 couches et 50 neurones par couche et  $\text{num\_domain} = 1000$ ,  $\text{num\_boundary} = 100$ ,  $\text{num\_initial} = 100$ , nous obtenons un résultat avec l'erreur acceptable. Et nous avons des implémentations graphiques suivantes :

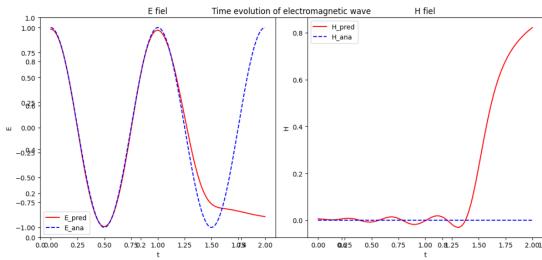


FIGURE 15 – Résultat optimal

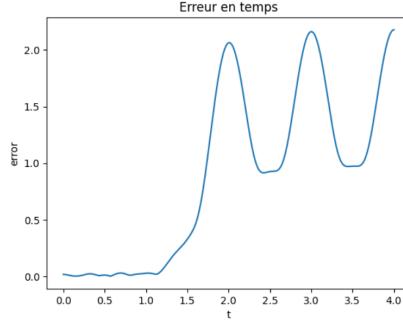


FIGURE 16 – La courbe de perte du modèle pendant l’entraînement

Nombre d'époques	Erreur maximale
5000	0.039425258
10000	0.02372258
15000	0.01266445
20000	0.00871603

TABLE 3 – Évolution de l’erreur maximale en fonction du nombre d’époques.

Nous constatons que l’erreur diminue de manière significative jusqu’à 6000 époques, après quoi l’amélioration devient marginale.

Étape	Train Loss	Test Loss	Test Metric	Temps d’entraînement (s)
5000	1.66e-04	2.07e-04	[1.42e-02]	85.709756
10000	3.58e-04	3.79e-04	[9.04e-03]	166.003110
15000	6.51e-05	7.52e-05	[7.00e-03]	254.184212
20000	2.33e-05	3.26e-05	[6.07e-03]	335.934791

TABLE 4 – Évolution des performances du modèle au fil des époques

## 2 Problème 2 :Propagation d’impulsion dans un domaine périodique homogène

### 2.1 Contexte

Nous considérons les équations de Maxwell en 1D dans le domaine temporel :

$$\varepsilon_r \frac{\partial E(x, t)}{\partial t} - \frac{\partial H(x, t)}{\partial x} = 0, \quad (5)$$

$$\mu_r \frac{\partial H(x, t)}{\partial t} - \frac{\partial E(x, t)}{\partial x} = 0. \quad (6)$$

On définit  $\Omega = [0, 6]$ .

Dans la suite du travail, on prendra  $\epsilon_r(x) = \mu_r(x) = 1, \forall x \in \Omega$

Les conditions aux limites sont définies par :

$$E(x=0, t) = E(x=6, t), H(x=0, t) = H(x=6, t) \quad \forall t \in [0, T_f]$$

Les conditions initiales sont données par :

$$E(x, 0) = e^{-\alpha(x-x_g)^2} \tag{7}$$

$$H(x, 0) = -e^{-\alpha(x-x_g)^2}, \tag{8}$$

Dans la suite du travail, on prendra par exemple  $\alpha = 10$  et  $T_f = 4$ .

## 2.2 Solutions numériques obtenue par PINNs

Nous avons obtenu par PINNs une solution numérique implémentée par le figure suivant :

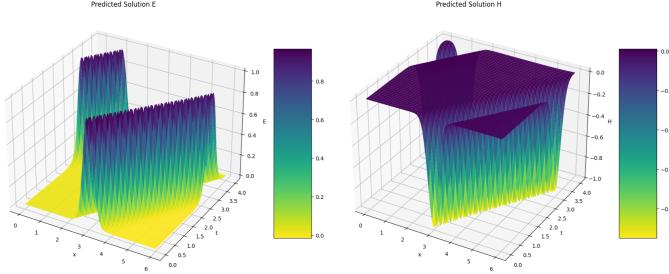


FIGURE 17 – Solution globale

Nous observons l'évolution d'un point unique dans l'espace au fil du temps, et nous obtenons le résultat suivant :

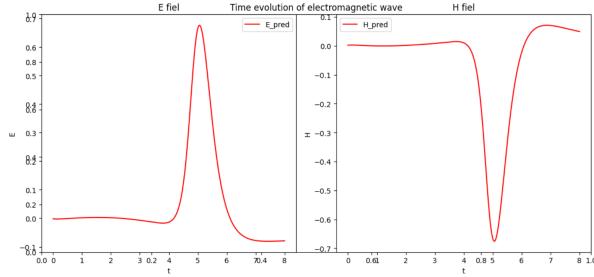


FIGURE 18 – Variation d'un point au cours du temps

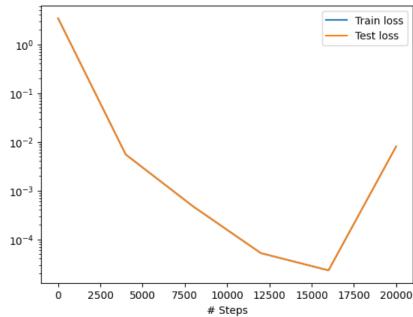


FIGURE 19 – La courbe de perte du modèle pendant l'entraînement

D'après le graphique, à mesure que le nombre de pas d'entraînement augmente, tant la perte d'entraînement que la perte de test diminuent, ce qui indique

que le modèle converge pour les données d'entraînement et de test. Vers 15000 pas, la perte de test atteint un minimum, puis augmente légèrement, suggérant que le modèle y atteint une bonne performance de généralisation et qu'un entraînement supplémentaire pourrait entraîner un surapprentissage.

## Conclusion

L'augmentation des échantillons frontaliers améliore la stabilité des prédictions, tandis que l'accroissement des échantillons de domaine influe directement sur la précision globale. Par ailleurs, la performance du modèle s'améliore de manière exponentielle avec l'augmentation du nombre d'époques. De plus, accroître le nombre de couches et de neurones par couche améliore progressivement la précision des prédictions tout en réduisant les erreurs. Toutefois, une complexification excessive du modèle, bien qu'elle puisse renforcer la précision, risque d'allonger le temps d'entraînement et d'augmenter le surapprentissage. Il est donc crucial de trouver un équilibre optimal entre précision, ressources de calcul et efficacité lors du choix du modèle.

### 3 Annexes : Codes pour la mise en oeuvre

#### 3.1 Annexe 1 : mise en oeuvre du problème 1

```
# ***Charge deepxde***
!pip install deepxde
# ***Problem definition***
import deepxde as dde
import numpy as np
import matplotlib.pyplot as plt

m = 1 # par défaut
#m = 2
omega = m * np.pi

geom = dde.geometry.Interval(0, 1)
timedomain = dde.geometry.TimeDomain(0, 4) # for m =1, the problem is 2 periodic
geomtime = dde.geometry.GeometryXTime(geom, timedomain)

def pde(x, y):
    E, H = y[:, 0:1], y[:, 1:2]
    dE_t = dde.grad.jacobian(y, x, i=0, j=1)
    dH_x = dde.grad.jacobian(y, x, i=1, j=0)
    dH_t = dde.grad.jacobian(y, x, i=1, j=1)
    dE_x = dde.grad.jacobian(y, x, i=0, j=0)

    eq1 = dE_t - dH_x
    eq2 = dH_t - dE_x
    return [eq1, eq2]

# boudary : Parois métalliques parfaites E(0,t) = E(1,t) = 0
def boundary_E(x, on_boundary):
    return on_boundary and (np.isclose(x[0], 0) or np.isclose(x[0], 1))

bc_E = dde.DirichletBC(geomtime, lambda x: 0, boundary_E, component=0)

# initial (E(x,0) = sin(m*pi*x), H(x,0) = 0)
def initial_E(x):
    return np.sin(m * np.pi * x[:, 0:1])

def initial_H(x):
```

```

        return np.zeros_like(x[:, 0:1])

ic_E = dde.IC(geomtime, initial_E, lambda _, on_initial: on_initial, component=0)
ic_H = dde.IC(geomtime, initial_H, lambda _, on_initial: on_initial, component=1)

# Analytical solution
def analy_sol(x):
    E = np.sin(m * np.pi * x[:, 0:1]) * np.cos(omega * x[:, 1:2])
    H = np.cos(m * np.pi * x[:, 0:1]) * np.sin(omega * x[:, 1:2])
    return np.hstack((E, H))

# ***Part1: Epochs Trainning***
# Trainning1: 4 layers and 50 neurons epochs = 1000*

layer_size = [2] + [50] * 4 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,
    num_boundary=100,
    num_initial=100,
    num_test=300,
    solution = analy_sol
)

# Define and train model
model = dde.Model(data, net)
optimizer = "adam"
model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
losshistory, train_state = model.train(epochs=1000, display_every=500)

# ***Time evaluation***
# visualization
T = 2 * np.pi / omega
x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

```

```

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")
plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
## Plot Error
error = []
x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instant, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

    E_ana = analy_sol(x)[0, 0:1]
    H_ana = analy_sol(x)[0, 1:2]

    error_E = np.abs(E_field - E_ana) # erreur sur champs électriques
    error_H = np.abs(H_field - H_ana) # erreur sur champs magnétiques
    error.append(np.sqrt(error_E**2 + error_H**2))
    #print(error)

plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Erreur en temps')
plt.show()

```

```

print(f"error_max in the training domain = {max(error[0:200])}")
# Traninng2: 4 layers and 50 neuronsepochs = 3000*
layer_size = [2] + [50] * 4 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,
    num_boundary=100,
    num_initial=100,
    num_test=300,
    solution = analy_sol
)

# Define and train model
model = dde.Model(data, net)
optimizer = "adam"
model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
losshistory, train_state = model.train(epochs=3000, display_every=500)
# ***Time Evaluation***
# visualization
T = 2 * np.pi / omega
x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

```

```

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")
plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
## Plot Error
error = []
x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instanciation, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

    E_ana = analy_sol(x)[0, 0:1]
    H_ana = analy_sol(x)[0, 1:2]

    error_E = np.abs(E_field - E_ana) # erreur sur champs électriques
    error_H = np.abs(H_field - H_ana) # erreur sur champs magnétiques
    error.append(np.sqrt(error_E**2 + error_H**2))
    #print(error)
plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Erreur en temps')
plt.show()
print(f"error_max in the training domain = {max(error[0:200])}")
# Training 3: 4 layers and 50 neurons per epoch = 6000
layer_size = [2] + [50] * 4 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,
    num_boundary=100,
)

```

```

        num_initial=100,
        num_test=300,
        solution = analy_sol
    )

    # Define and train model
    model = dde.Model(data, net)
    optimizer = "adam"
    model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
    losshistory, train_state = model.train(epochs=6000, display_every=500)
    # ***Time Evaluation***
    # visualization
    T = 2 * np.pi / omega
    x_value = 0.25 # fixed point
    x2 = np.linspace(0, 2 * T, 400)
    x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
    y = model.predict(x)

    E_field = y[:, 0:1]
    H_field = y[:, 1:2]

    E_ana = analy_sol(x)[:, 0:1]
    H_ana = analy_sol(x)[:, 1:2]

    plt.figure(figsize=(12, 6))
    plt.title("Time evolution of electromagnetic wave")
    plt.subplot(1, 2, 1)
    plt.plot(x2, E_field, 'r', label="E_pred")
    plt.plot(x2, E_ana, 'b--', label="E_ana")
    plt.xlabel("t")
    plt.ylabel("E")
    plt.title('E field')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(x2, H_field, 'r', label="H_pred")
    plt.plot(x2, H_ana, 'b--', label="H_ana")
    plt.xlabel("t")
    plt.ylabel("H")
    plt.title('H field')
    plt.legend()

    plt.tight_layout()
    plt.show()
    ## Plot Error
    error = []

```

```

x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instant, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

    E_ana = analy_sol(x)[0, 0:1]
    H_ana = analy_sol(x)[0, 1:2]

    error_E = np.abs(E_field - E_ana) # erreur sur champs electriques
    error_H = np.abs(H_field - H_ana) # erreur sur champs magntiques
    error.append(np.sqrt(error_E**2 + error_H**2))
    #print(error)
plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Erreur en temps')
plt.show()
print(f"error_max in the trainning domain = {max(error[0:200])}")
# Trainning 4: 4 layers and 50 neuronsepochs = 12000
layer_size = [2] + [50] * 4 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,
    num_boundary=100,
    num_initial=100,
    num_test=300,
    solution = analy_sol
)

# Define and train model
model = dde.Model(data, net)
optimizer = "adam"
model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
losshistory, train_state = model.train(epochs=12000, display_every=1000)

# ***Time Evaluation***
# visualization

```

```

T = 2 * np.pi / omega
x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")
plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
## Plot Error
error = []
x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instant, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

    E_ana = analy_sol(x)[0, 0:1]
    H_ana = analy_sol(x)[0, 1:2]

    error_E = np.abs(E_field - E_ana) # erreur sur champs electriques
    error_H = np.abs(H_field - H_ana) # erreur sur champs magntiques

```

```

    error.append(np.sqrt(error_E**2 + error_H**2))
    #print(error)
plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Erreurs en temps')
plt.show()
print(f"error_max dans le domaine d'entraînement = {max(error[0:200])}")

# Plot result
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(0, 1, 100)
t = np.linspace(0, 1, 100)
X, T = np.meshgrid(x, t)

E_ex, H_ex = analy_sol(np.hstack((X.reshape(-1, 1), T.reshape(-1, 1))).T
E_ex = E_ex.reshape(X.shape)
H_ex = H_ex.reshape(X.shape)

E_pred, H_pred = model.predict(np.hstack((X.reshape(-1, 1), T.reshape(-1, 1))).T
E_pred = E_pred.reshape(X.shape)
H_pred = H_pred.reshape(X.shape)

fig = plt.figure(figsize=(18, 10))

# E
ax1 = fig.add_subplot(221, projection='3d')
E_ex = ax1.plot_surface(X, T, E_ex, cmap='viridis_r', edgecolor='none')
ax1.set_title('Analytical Solution E')
ax1.set_xlabel('x')
ax1.set_ylabel('t')
ax1.set_zlabel('E')
fig.colorbar(E_ex, ax=ax1, shrink=0.5, aspect=5)

# H
ax2 = fig.add_subplot(222, projection='3d')
H_ex = ax2.plot_surface(X, T, H_ex, cmap='viridis_r', edgecolor='none')
ax2.set_title('Analytical Solution H')
ax2.set_xlabel('x')
ax2.set_ylabel('t')
ax2.set_zlabel('H')
fig.colorbar(H_ex, ax=ax2, shrink=0.5, aspect=5)

```

```

# prediction of E
ax3 = fig.add_subplot(223, projection='3d')
E_pred = ax3.plot_surface(X, T, E_pred, cmap='viridis_r', edgecolor='none')
ax3.set_title('Predicted Solution E')
ax3.set_xlabel('x')
ax3.set_ylabel('t')
ax3.set_zlabel('E')
fig.colorbar(E_pred, ax=ax3, shrink=0.5, aspect=5)
# prediction of H
ax4 = fig.add_subplot(224, projection='3d')
H_pred = ax4.plot_surface(X, T, H_pred, cmap='viridis_r', edgecolor='none')
ax4.set_title('Predicted Solution H')
ax4.set_xlabel('x')
ax4.set_ylabel('t')
ax4.set_zlabel('H')
fig.colorbar(H_pred, ax=ax4, shrink=0.5, aspect=5)

plt.tight_layout()
plt.show()

# visualization
T = 2 * np.pi / omega
x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")

```

```

plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
# ***Part 2: Neuron Network Trainning***
# Trainning 1: 6000 epochs, 2 layers and 20 Nerons
layer_size = [2] + [20] * 2 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,
    num_boundary=100,
    num_initial=100,
    num_test=300,
    solution = analy_sol
)

# Define and train model
model = dde.Model(data, net)
optimizer = "adam"
model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
losshistory, train_state = model.train(epochs=6000, display_every=500)
# ***Time Evaluation***
# visualization
T = 2 * np.pi / omega
x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")

```

```

plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")
plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
## Plot error
error = []
x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instant, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

    E_ana = analy_sol(x)[0, 0:1]
    H_ana = analy_sol(x)[0, 1:2]

    error_E = np.abs(E_field - E_ana) # erreur sur champs electriques
    error_H = np.abs(H_field - H_ana) # erreur sur champs magnétiques
    error.append(np.sqrt(error_E**2 + error_H**2))
    #print(error)
plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Error in time')
plt.show()
print(f"error_max in the training domain = {max(error[0:200])}")
# Training 2 : 6000 epochs, 2 layers and 50 Neurons
layer_size = [2] + [50] * 2 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

```

```

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,
    num_boundary=100,
    num_initial=100,
    num_test=300,
    solution = analy_sol
)

# Define and train model
model = dde.Model(data, net)
optimizer = "adam"
model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
losshistory, train_state = model.train(epochs=6000, display_every=500)
# ***Time Evaluation***
# visualization
T = 2 * np.pi / omega
x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")
plt.xlabel("t")
plt.ylabel("H")

```

```

plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
## Plot error
error = []
x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instant, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

    E_ana = analy_sol(x)[0, 0:1]
    H_ana = analy_sol(x)[0, 1:2]

    error_E = np.abs(E_field - E_ana) # erreur sur champs electriques
    error_H = np.abs(H_field - H_ana) # erreur sur champs magnétiques
    error.append(np.sqrt(error_E**2 + error_H**2))
    #print(error)
plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Erreur en temps')
plt.show()
print(f"error_max in the training domain = {max(error[0:200])}")
# ***Training 3 : 6000 epochs, 3 layers and 20 Neurons***
layer_size = [2] + [20] * 3 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,
    num_boundary=100,
    num_initial=100,
    num_test=300,
    solution = analy_sol
)

# Define and train model

```

```

model = dde.Model(data, net)
optimizer = "adam"
model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
losshistory, train_state = model.train(epochs=6000, display_every=500)
# ***Time Evaluation***
# visualization
T = 2 * np.pi / omega
x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")
plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
## Plot error
error = []
x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instant, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

```

```

E_ana = analy_sol(x)[0, 0:1]
H_ana = analy_sol(x)[0, 1:2]

error_E = np.abs(E_field - E_ana) # erreur sur champs electriques
error_H = np.abs(H_field - H_ana) # erreur sur champs magnétiques
error.append(np.sqrt(error_E**2 + error_H**2))
#print(error)
plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Erreur en temps')
plt.show()
print(f"error_max in the training domain = {max(error[0:200])}")
# ***Trainning 4 : 6000 epochs, 3 layers and 50 Neurons***
layer_size = [2] + [50] * 3 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,
    num_boundary=100,
    num_initial=100,
    num_test=300,
    solution = analy_sol
)

# Define and train model
model = dde.Model(data, net)
optimizer = "adam"
model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
losshistory, train_state = model.train(epochs=6000, display_every=500)
# ***Time Evaluation***
# visualization
T = 2 * np.pi / omega
x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]

```

```

H_field = y[:, 1:2]

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")
plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
## Plot error
error = []
x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instant, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

    E_ana = analy_sol(x)[0, 0:1]
    H_ana = analy_sol(x)[0, 1:2]

    error_E = np.abs(E_field - E_ana) # erreur sur champs electriques
    error_H = np.abs(H_field - H_ana) # erreur sur champs magntiques
    error.append(np.sqrt(error_E**2 + error_H**2))
    #print(error)
plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Erreur en temps')

```

```

plt.show()
print(f"error_max in the trainning domain = {max(error[0:200])}")
# ***Trainning 5 : 6000 epochs, 4 layers and 20 Nerons***
layer_size = [2] + [20] * 4 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,
    num_boundary=100,
    num_initial=100,
    num_test=300,
    solution = analy_sol
)

# Define and train model
model = dde.Model(data, net)
optimizer = "adam"
model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
losshistory, train_state = model.train(epochs=6000, display_every=500)
# ***Time Evaluation***
# visualization
T = 2 * np.pi / omega
x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')

```

```

plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")
plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
## Plot error
error = []
x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instant, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

    E_ana = analy_sol(x)[0, 0:1]
    H_ana = analy_sol(x)[0, 1:2]

    error_E = np.abs(E_field - E_ana) # erreur sur champs electriques
    error_H = np.abs(H_field - H_ana) # erreur sur champs magnétiques
    error.append(np.sqrt(error_E**2 + error_H**2))
    #print(error)
plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Erreur en temps')
plt.show()
print(f"error_max in the training domain = {max(error[0:200])}")
# ***Training 6 : 6000 epochs, 4 layers and 50 Neurons***
layer_size = [2] + [50] * 4 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,

```

```

        num_boundary=100,
        num_initial=100,
        num_test=300,
        solution = analy_sol
    )

# Define and train model
model = dde.Model(data, net)
optimizer = "adam"
model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
losshistory, train_state = model.train(epochs=6000, display_every=500)
# ***Time Evaluation***
# visualization
T = 2 * np.pi / omega
x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")
plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
## Plot error

```

```

error = []
x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instant, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

    E_ana = analy_sol(x)[0, 0:1]
    H_ana = analy_sol(x)[0, 1:2]

    error_E = np.abs(E_field - E_ana) # erreur sur champs electriques
    error_H = np.abs(H_field - H_ana) # erreur sur champs magnétiques
    error.append(np.sqrt(error_E**2 + error_H**2))
    #print(error)
plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Erreur en temps')
plt.show()
print(f"error_max in the training domain = {max(error[0:200])}")
# ***Trainning 7 : 6000 epochs, 4 layers and 80 Neurons***
layer_size = [2] + [80] * 4 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, ic_E, ic_H],
    num_domain=1000,
    num_boundary=100,
    num_initial=100,
    num_test=300,
    solution = analy_sol
)

# Define and train model
model = dde.Model(data, net)
optimizer = "adam"
model.compile(optimizer, lr=0.001, metrics=["l2 relative error"])
losshistory, train_state = model.train(epochs=6000, display_every=500)
# visualization
T = 2 * np.pi / omega

```

```

x_value = 0.25 # fixed point
x2 = np.linspace(0, 2 * T, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

E_ana = analy_sol(x)[:, 0:1]
H_ana = analy_sol(x)[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.plot(x2, E_ana, 'b--', label="E_ana")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.plot(x2, H_ana, 'b--', label="H_ana")
plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
error = []
x2 = np.linspace(0, 4, 400)
for t in x2: # a chaque l'instant, on calcule l'erreur
    x = np.column_stack([x_value, t])
    y = model.predict(x)
    E_field = y[0, 0:1]
    H_field = y[0, 1:2]

    E_ana = analy_sol(x)[0, 0:1]
    H_ana = analy_sol(x)[0, 1:2]

    error_E = np.abs(E_field - E_ana) # erreur sur champs electriques
    error_H = np.abs(H_field - H_ana) # erreur sur champs magntiques
    error.append(np.sqrt(error_E**2 + error_H**2))
#print(error)

```

```

plt.figure()
plt.plot(x2, error)
plt.xlabel("t")
plt.ylabel("error")
plt.title('Erreur en temps')
plt.show()
print(f"error_max in the training domain = {max(error[0:200])}")

```

### 3.2 Annexe 3 : mise en oeuvre du problème 2

```

# ***Charge deepxde***
!pip install deepxde
# ***Problem definition***
import deepxde as dde
import numpy as np
import matplotlib.pyplot as plt

alpha = 10.0
x_g = 3.0
Tf = 4.0
c = 1.0

# calculate domain
geom = dde.geometry.Interval(0, 6)
timedomain = dde.geometry.TimeDomain(0, Tf)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)

# PDE
def pde(x, y):
    E, H = y[:, 0:1], y[:, 1:2]
    dE_t = dde.grad.jacobian(y, x, i=0, j=1)
    dH_x = dde.grad.jacobian(y, x, i=1, j=0)
    dH_t = dde.grad.jacobian(y, x, i=1, j=1)
    dE_x = dde.grad.jacobian(y, x, i=0, j=0)
    return [dE_t - dH_x, dH_t - dE_x]

# boundary condition
def boundary_lr(x, on_boundary):
    return on_boundary and (np.isclose(x[0], 0) or np.isclose(x[0], 6))

bc_E = dde.icbc.PeriodicBC(

```

```

        geomtime,
        component_x = 0,
        derivative_order = 0,
        on_boundary=boundary_lr
    )

bc_H = dde.icbc.PeriodicBC(
    geomtime,
    component_x = 1,
    derivative_order = 0,
    on_boundary=boundary_lr
)

# initial condition
def gaussian(x):
    return np.exp(-alpha * (x[:, 0:1] - x_g)**2)

ic_E = dde.icbc.IC(
    geomtime,
    lambda x: gaussian(x),
    lambda _, on_initial: on_initial,
    component=0
)

ic_H = dde.icbc.IC(
    geomtime,
    lambda x: -gaussian(x),
    lambda _, on_initial: on_initial,
    component=1
)

# Neuron Network construction and training
layer_size = [2] + [50] * 6 + [2]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_E, bc_H, ic_E, ic_H],
    num_domain=1000,
    num_boundary=180,
    num_initial=180,
)

```

```

# Define and train model
model = dde.Model(data, net)
model.compile("adam", lr=0.001)
losshistory, train_state = model.train(iterations=20000, display_every=4000)
# ***Time Evaluation***
# visualization
x_value = 2 # fixed point
x2 = np.linspace(0, 2 * Tf, 400)
x = np.stack((np.full(x2.shape, x_value), x2), axis=-1)
y = model.predict(x)

E_field = y[:, 0:1]
H_field = y[:, 1:2]

plt.figure(figsize=(12, 6))
plt.title("Time evolution of electromagnetic wave")
plt.subplot(1, 2, 1)
plt.plot(x2, E_field, 'r', label="E_pred")
plt.xlabel("t")
plt.ylabel("E")
plt.title('E field')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x2, H_field, 'r', label="H_pred")
plt.xlabel("t")
plt.ylabel("H")
plt.title('H field')
plt.legend()

plt.tight_layout()
plt.show()
dde.saveplot(losshistory, train_state, issave=True, isplot=True)
# Electromagnetic fields plot
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(0, 6, 100)
t = np.linspace(0, 4, 100)
X, T = np.meshgrid(x, t)

E_pred, H_pred = model.predict(np.hstack((X.reshape(-1, 1), T.reshape(-1, 1)))).T
E_pred = E_pred.reshape(X.shape)
H_pred = H_pred.reshape(X.shape)

```

```

fig = plt.figure(figsize=(18, 10))

# prediction of E
ax3 = fig.add_subplot(121, projection='3d')
E_pred = ax3.plot_surface(X, T, E_pred, cmap='viridis_r', edgecolor='none')
ax3.set_title('Predicted Solution E')
ax3.set_xlabel('x')
ax3.set_ylabel('t')
ax3.set_zlabel('E')
fig.colorbar(E_pred, ax=ax3, shrink=0.5, aspect=5)
# prediction of H
ax4 = fig.add_subplot(122, projection='3d')
H_pred = ax4.plot_surface(X, T, H_pred, cmap='viridis_r', edgecolor='none')
ax4.set_title('Predicted Solution H')
ax4.set_xlabel('x')
ax4.set_ylabel('t')
ax4.set_zlabel('H')
fig.colorbar(H_pred, ax=ax4, shrink=0.5, aspect=5)

plt.tight_layout()
plt.show()

```