In [21]:

```python
#!/usr/bin/env python
# coding: utf-8

from matplotlib import cm
from sympy import Matrix
import sympy as sm
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
import numpy as np
import os
import json
from time import time

################## check tensorflow is using GPU ##################
from tensorflow.python.client import device_lib
physical_devices = tf.config.list_physical_devices("GPU")
print("Num GPUs Available: ", len(physical_devices))
print(device_lib.list_local_devices())
# what if empty...
tf.config.experimental.set_memory_growth(physical_devices[0], True)
# On windows systems you cannot install NCCL that is required for multi GPU
# So we need to follow hierarchical copy method or reduce to single GPU (less efficient than the former)
strategy = tf.distribute.MirroredStrategy(
    devices=['GPU:0'], cross_device_ops=tf.distribute.HierarchicalCopyAllReduce())

DTYPE = 'float32'

tf.keras.backend.set_floatx(DTYPE)

################## proper to the computer used ##################
__file__ = 'C:/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/Tsunami'

print('\ncwd:', os.getcwd())
os.chdir(__file__)
print('changed to:', os.getcwd(), '\n')


################## Create the model ##################

def generate_model(l_units, noise=False):
    # méthode API Sequential
    n_hidden = len(l_units)
    model = keras.models.Sequential([
        keras.layers.Input(shape=(2))
    ])
    if noise:
        model.add(keras.layers.GaussianNoise(stddev=1e-4))
    for i in range(n_hidden):
        model.add(keras.layers.Dense(
            l_units[i], activation='relu', kernel_initializer='he_normal'))
    model.add(keras.layers.Dense(1, use_bias=False))
    # use_bias=False otherwise returns error after
    # May be the cause of a stagnation in the validation ? (can circumvent by creating a layer class only addin
    model.summary()
    return model


def test_0():
    model = generate_model()

################## define the EDP to solve ##################
# Given EDO
def f(X):
    return tf.sin(np.pi*X[:, 0])*tf.sin(np.pi*X[:, 1])


def boundary_conditions(X):
    return 0


def residual(du_dxx, du_dyy, f_ind):
    return du_dxx+du_dyy+f_ind


def differentiate(model, x):
    with tf.GradientTape(persistent=True) as tape:
        x1, x2 = x[:, 0:1], x[:, 1:2]
        tape.watch(x1)
        tape.watch(x2)
        u = model(tf.stack([x1[:, 0], x2[:, 0]], axis=1))
        du_dx = tape.gradient(u, x1)
        du_dy = tape.gradient(u, x2)
    du_dxx = tape.gradient(du_dx, x1)
    du_dyy = tape.gradient(du_dy, x2)
    return du_dx, du_dxx, du_dy, du_dyy



################## mesh the domain ##################
grid_length = 1000

X = np.linspace(0, 1, grid_length, endpoint=True)
Y = np.linspace(0, 1, grid_length, endpoint=True)
tf_coords = tf.convert_to_tensor(
    [tf.constant([x, y], dtype=DTYPE) for x in X for y in Y])
tf_boundary_coords = tf.convert_to_tensor([tf.constant([x, y], dtype=DTYPE) for x in [
                                          0, 1] for y in Y] + [tf.constant([x, y], dtype=DTYPE) for y in [0, 1]


##########################################################################################
################## Method 3: g automatically respects the boundary conditions ########################
# article : 1997_Artificial_neural_networks_for_solving_ordinary_and_partial_differential_equations.pdf

################## Set F here ##################
# Dummy F


x, y = sm.symbols('x,y')


def expr_dummy_F():
    return x*(1-x)*y*(1-y)


expr_F = expr_dummy_F()
dexpr_F_dx = sm.diff(expr_F, x, 1)
dexpr_F_dxx = sm.diff(dexpr_F_dx, x, 1)
dexpr_F_dy = sm.diff(expr_F, y, 1)
dexpr_F_dyy = sm.diff(dexpr_F_dy, y, 1)


# remark: You can forget a no lambdified expression => here we greatly avoid 'for' loops

expr_F = sm.lambdify([x, y], Matrix([expr_F]), 'numpy')
dexpr_F_dx = sm.lambdify([x, y], Matrix([dexpr_F_dx]), 'numpy')
dexpr_F_dxx = sm.lambdify([x, y], Matrix([dexpr_F_dxx]), 'numpy')
dexpr_F_dy = sm.lambdify([x, y], Matrix([dexpr_F_dy]), 'numpy')
dexpr_F_dyy = sm.lambdify([x, y], Matrix([dexpr_F_dyy]), 'numpy')


def evaluate_F_and_diff(X):
    F = tf.squeeze(tf.transpose(expr_F(X[:, 0], X[:, 1])), axis=-1)
    dF_dx = tf.expand_dims(dexpr_F_dx(X[:, 0], X[:, 1]), axis=-1)
    dF_dxx = tf.expand_dims(dexpr_F_dxx(X[:, 0], X[:, 1]), axis=-1)
    dF_dy = tf.expand_dims(dexpr_F_dy(X[:, 0], X[:, 1]), axis=-1)
    dF_dyy = tf.expand_dims(dexpr_F_dyy(X[:, 0], X[:, 1]), axis=-1)

    return F, dF_dx, dF_dxx, dF_dy, dF_dyy

 # oddly enough expr_F and dexpr_F_d... do not have the same output


# # #### F of F_functions


# frontier_coords = Pstud._set_coords_rectangle(1, 1, 10)

# l_orders = [(1, 0), (2, 0), (0, 1), (0, 2)]
# strfn = 'sinxpy_real'
# F = F2D(frontier_coords, strfn, l_orders=l_orders)

# # prepare to infer on large matrices :
# F.expr = sm.lambdify(F.variables, Matrix([F.expr]), 'numpy')
# for t_order in l_orders:
#     F.reduced_tab_diff[F.dico_order_to_index[t_order]] = sm.lambdify(
#         F.variables, F.reduced_tab_diff[F.dico_order_to_index[t_order]], 'numpy')


# def evaluate_F_and_diff(X):
#     '''
#     evaluate F and its differentiates get in F.reduced_tab_diff
#     Variables:
#     -X: an array or tensor tf of the coordinates

#     Returns:
#     -l_eval: list of the evaluations. To know which element corresponds to which order, use F.dico_order_to_i

#     remark: to add to F2D class
#     '''
#     l_eval = [tf.squeeze(tf.transpose(F.expr(X[:, 0], X[:, 1])), axis=-1)]

#     for i, t_order in enumerate(F.reduced_tab_diff):
#         l_eval.append(tf.expand_dims(
#             F.reduced_tab_diff[i](X[:, 0], X[:, 1]), axis=-1))

#     return l_eval



################## Set A here ##################

A = 0
dA_dxx = 0
dA_dyy = 0
```

```
Num GPUs Available:  1
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 3961130884003023103
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 5730467840
locality {
  bus_id: 1
  links {
  }
}
incarnation: 4587379600596016239
physical_device_desc: "device: 0, name: NVIDIA GeForce RTX 3080 Laptop GPU, pci bus id: 0000:01:00.0, compute c
apability: 8.6"
xla_global_id: 416903419
]
INFO:tensorflow:Using MirroredStrategy with devices ('/job:localhost/replica:0/task:0/device:GPU:0',)

cwd: C:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami
changed to: C:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami
```

In [22]:

```python
def try_config(config,id_add):
    '''
    A run of the full algorithm described in the paper of 1997.
    Variables:
    -config (dict): The hyperparameters used to construct the model and set the training loop are in config.
    -id_add (int): add id_add to the trial_id to avoid overwriting previous trials
    '''
    print('config:\n',config)
    config_model = config['config_model']
    config_training = config['config_training']

    l_units = config_model['l_units']
    noise = config_model['noise']
    learning_rate = config_model['learning_rate']
    if config_model['optimizer'] == "Adam":
        optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    else:
        print('optimizer is not known !')

    learning_rate = config_model['learning_rate']


    # generate model
    model = generate_model(l_units, noise=noise)


    # Universal Approximator
    # @tf.function  # TODO: learn to use it to accelerate the computations
    def g_3(X, training=True):
        # F_x = Pstud._eval_polynome_numpy(F_xpy_real,x[0,0],x[0,1])
        N_X = model(X, training=training)
        return tf.squeeze(tf.transpose(expr_F(X[:, 0], X[:, 1])), axis=-1)*N_X


    # Custom loss function to approximate the derivatives

    def custom_loss_3(tf_sample_coords):
        dN_dx, dN_dxx, dN_dy, dN_dyy = differentiate(
            model, tf_sample_coords)
        f_r = tf.reshape(f(tf_sample_coords), [batch_size, 1])

        F, dF_dx, dF_dxx, dF_dy, dF_dyy = evaluate_F_and_diff(tf_sample_coords)

        dg_dxx = dF_dxx + 2*dF_dx*dN_dx + F*dN_dxx + dA_dxx
        dg_dyy = dF_dyy + 2*dF_dy*dN_dy + F*dN_dyy + dA_dyy
        res = residual(dg_dxx, dg_dyy, f_r)

        loss = tf.reduce_mean(tf.square(res))
        return loss

    # train of method 3:

    def train_step_3(tf_sample_coords):
        with tf.GradientTape() as tape:
            loss = custom_loss_3(tf_sample_coords)
        trainable_variables = model.trainable_variables
        gradients = tape.gradient(loss, trainable_variables)
        optimizer.apply_gradients(zip(gradients, trainable_variables))
        return loss


    mae_metric = tf.keras.metrics.MeanAbsoluteError(
        name="mean_absolute_error", dtype=None)


    def validate(validation_coords):
        _, dg_dxx, _, dg_dyy = differentiate(g_3, validation_coords)
        f_r = tf.reshape(f(validation_coords), [tf.shape(validation_coords)[0], 1])
        res = residual(dg_dxx, dg_dyy, f_r)
        val_mae = mae_metric(res, tf.zeros(tf.shape(res))).numpy()
        return val_mae


    # Training the Model of method 3:


    trial_id = config['trial_id']+id_add
    epochs_max = config_training['epochs_max']
    n_trains = config_training['n_trains']
    batch_size = config_training['batch_size']
    display_step = config_training['display_step']
    tol = config_training['tol']
    patience = config_training['patience']



    ## !!! to change according to the way the folders are arranged
    folder_dir = f'differentiate/hypertuning/byHand/trial_{trial_id}/'



    # TODO: learn how to use Yaml instead... (piece of advice from Jules S.)

    if not(os.path.exists(folder_dir)):
        os.mkdir(folder_dir)
    with open(folder_dir+f'config_{trial_id}.json', 'w') as fp:
        json.dump(config, fp, indent=4)



    history = {'mean_loss': [], 'val_mae': []}

    epoch = 0
    val_mae = np.infty
    val_mae_reached = (val_mae <= tol)
    EarlyStopped = False

    # tf.keras.backend.set_learning_phase(1) # 'globally' activate training mode, slightly too strong maybe : c
    while not(EarlyStopped) and not(val_mae_reached) and epoch < epochs_max:
        epoch += 1
        time_start = time()
        print('epoch:', epoch, end=' ')
        losses = []

        indices = np.random.randint(tf_coords.shape[0], size=batch_size)
        tf_sample_coords = tf.convert_to_tensor([tf_coords[i] for i in indices])
        for k in range(n_trains):
            if k % display_step == display_step-1:
                print('.', end='')
            losses.append(train_step_3(tf_sample_coords))
        loss = np.mean(losses)

        # create validation_coords
        indices = np.random.randint(tf_coords.shape[0], size=100)
        tf_val_coords = tf.convert_to_tensor([tf_coords[i] for i in indices])
        tf_val_coords = tf_val_coords + \
            tf.random.normal(shape=tf.shape(
                tf_val_coords).numpy(), mean=0, stddev=1e-3)
        val_mae = validate(tf_val_coords)

        print("mean_loss:", loss, end=' ')
        print('val_mae:', val_mae, end=' ')
        history['mean_loss'].append(loss)
        history['val_mae'].append(val_mae)

        # time_end_training = time()
        # print('duration training :', time_end_training-time_start, end=' ')

        val_mae_reached = (val_mae <= tol)

        if val_mae_reached:
            print(f'\n tolerance set is reached : {val_mae}<={tol}')

        model.save(
            folder_dir+f'model_poisson_trial_{trial_id}_epoch_{epoch}_val_mae_{val_mae:6f}.h5')

        if (len(history['val_mae']) >= patience+1) and np.argmin(history['val_mae'][-(patience+1):]) == 0:
            print('\n EarlyStopping activated', end = ' ')
            EarlyStopped = True


        elif (len(history['val_mae']) >= patience+1):
            # clean the savings folder
            r_val_mae_epoch = epoch-patience
            r_val_mae = history['val_mae'][-(patience+1)]
            file_path = folder_dir + \
                f'model_poisson_trial_{trial_id}_epoch_{r_val_mae_epoch}_val_mae_{r_val_mae:6f}.h5'

            if os.path.exists(file_path):
                os.remove(file_path)
            else:
                print(file_path)
                print("The system cannot find the file specified")

        time_end_epoch = time()
        duration_epoch = time_end_epoch-time_start
        print('duration epoch:', duration_epoch)
        print()


    # not optimized
    min_val_mae = np.min(history['val_mae'])
    min_val_mae_epoch = np.argmin(history['val_mae'])+1


    model = keras.models.load_model(folder_dir+f'model_poisson_trial_{trial_id}_epoch_{min_val_mae_epoch}_val_m
    os.rename(folder_dir+f'model_poisson_trial_{trial_id}_epoch_{min_val_mae_epoch}_val_mae_{min_val_mae:6f}.h5
    print('best model loaded and renamed')

    # tf.keras.backend.set_learning_phase(0) # 'globally' disable training mode
    print("val_mae>tol:", val_mae > tol)

    plt.plot(np.arange(0, epoch), history['mean_loss'], label='mean_loss')
    plt.plot(np.arange(0, epoch), history['val_mae'], label='val_mae')
    # plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.title(
        f'epochs_max = {epochs_max},n_trains={n_trains},batch_size={batch_size}')
    plt.legend()
    plt.savefig(folder_dir+f"history_trial_{trial_id}.png",transparent=False)
    print(folder_dir+f"history_trial_{trial_id}.png")

    plt.plot(list(range(0, len(history['val_mae']))), history['val_mae'])
    plt.show()

    print(np.min(history['val_mae']))


# Does not learn with F = F2D(..., 'sinxpy_real') or 'xpy_real'
# - take a look at dF_dx and so on
# - ... at the hyperparameters
# -




# # Questions

# Quelle architecture ?
#
# Comment éviter l'overfitting ?
#
# Comment exploiter les avantages de l'IA ?
#
# Choix de l'optimizer + regularizer ? + Implémentation ?
#
# Implémentation de système d'EDP à plusieurs inconnues (étant des fonctions bien sûr) ? (Est-ce que c'est util
#
# Plus rapide ? Comment enlever les boucles `for` ? => mini_batch_gradient_descent ? done
#
# Besoin de batch_normalization ? + autres hyperparamètres ?

# # Idées

# Ajout de bruit en entrée contre l'overfitting
#
# Une sortie par inconnue

# %%
```

In [ ]:

```python
# a few train parameters to adjust
# use learning_rate = 1e-2, batch_size = 1000 for dummy F
###############
def generate_random_config(trial_id,grid_length):
    n_hidden_layers = np.random.randint(2,7)
    l_units = [5*np.random.randint(1,7) for _ in range(n_hidden_layers)]
    noise = np.random.randint(2)

    config_model = {
        'l_units': l_units,
        'noise': noise,
        'learning_rate': 1e-2,
        'optimizer': "Adam"
    }


    n_trains = 50*np.random.randint(2,5)
    config_training = {
        "epochs_max": 5000,
        "n_trains": n_trains,
        "batch_size": 8192,
        "display_step": 10,
        "tol": 1e-6,
        "patience": 50
    }

    config = {
        "trial_id": trial_id,
        "grid_length":grid_length, # do not change anything, here to inform
        "config_training": config_training,
        "config_model": config_model
    }

    return config

max_trials = 100
id_add=100
def randomTuning(max_trials,id_add):
    for trial_id in range(max_trials):
        config = generate_random_config(trial_id,grid_length)
        try_config(config,id_add=id_add)

randomTuning(max_trials,id_add) ############
```

In [26]:

```python
config_model = {
        'l_units': [30,30],
        'noise': 0,
        'learning_rate': 1e-4,
        'optimizer': "Adam"
    }


config_training = {
        "epochs_max": 5000,
        "n_trains": 10000,
        "batch_size": 65536,
        "display_step": 100,
        "tol": 1e-6,
        "patience": 5000
    }

config = {
        "trial_id": 0,
        "grid_length":grid_length,
        'remark':'do overfit',
        "config_training": config_training,
        "config_model": config_model
    }

try_config(config,id_add=100)
```

```
config:
 {'trial_id': 0, 'grid_length': 1000, 'remark': 'do overfit', 'config_training': {'epochs_max': 5000, 'n_train
s': 10000, 'batch_size': 65536, 'display_step': 100, 'tol': 1e-06, 'patience': 5000}, 'config_model': {'l_unit
s': [30, 30], 'noise': 0, 'learning_rate': 0.0001, 'optimizer': 'Adam'}}
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_42 (Dense)            (None, 30)                90

 dense_43 (Dense)            (None, 30)                930

 dense_44 (Dense)            (None, 1)                 30

=================================================================
Total params: 1,050
Trainable params: 1,050
Non-trainable params: 0
_________________________________________________________________
epoch: 1 ....................................................................................................me
an_loss: 0.01926151 val_mae: 0.40817046 WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
duration epoch: 231.39247179031372

epoch: 2 ....................................................................................................me
an_loss: 0.020836184 val_mae: 0.40400857 WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
duration epoch: 231.87281036376953

epoch: 3 ....................................................................................................me
an_loss: 0.013686223 val_mae: 0.40738142 WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
duration epoch: 244.179349899292

epoch: 4 ....................................................................................................me
an_loss: 0.013381184 val_mae: 0.4116396 WARNING:tensorflow:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
duration epoch: 247.49915432929993

epoch: 5
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
c:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami\differentiate\NN_method_3.ipy
nb Cell 4' in <cell line: 26>()
      <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=8'>9</a> config_training = {
     <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=9'>10</a>         "epochs_max": 5000,
     <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=10'>11</a>         "n_trains": 10000,
   (...)
     <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=14'>15</a>         "patience": 5000
     <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=15'>16</a>     }
     <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=17'>18</a> config = {
     <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=18'>19</a>         "trial_id": 0,
     <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=19'>20</a>         "grid_length":grid_length,
   (...)
     <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=22'>23</a>         "config_model": config_model
     <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=23'>24</a>     }
---> <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance
4/Tsunami/differentiate/NN_method_3.ipynb#ch0000003?line=25'>26</a> try_config(config,id_add=100)

c:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami\differentiate\NN_method_3.ipy
nb Cell 2' in try_config(config, id_add)
    <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/
Tsunami/differentiate/NN_method_3.ipynb#ch0000001?line=112'>113</a> losses = []
    <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/
Tsunami/differentiate/NN_method_3.ipynb#ch0000001?line=114'>115</a> indices = np.random.randint(tf_coords.shape
[0], size=batch_size)
--> <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/
Tsunami/differentiate/NN_method_3.ipynb#ch0000001?line=115'>116</a> tf_sample_coords = tf.convert_to_tensor([tf
_coords[i] for i in indices])
    <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/
Tsunami/differentiate/NN_method_3.ipynb#ch0000001?line=116'>117</a> for k in range(n_trains):
    <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/
Tsunami/differentiate/NN_method_3.ipynb#ch0000001?line=117'>118</a>     if k % display_step == display_step-1:

c:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami\differentiate\NN_method_3.ipy
nb Cell 2' in <listcomp>(.0)
    <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/
Tsunami/differentiate/NN_method_3.ipynb#ch0000001?line=112'>113</a> losses = []
    <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/
Tsunami/differentiate/NN_method_3.ipynb#ch0000001?line=114'>115</a> indices = np.random.randint(tf_coords.shape
[0], size=batch_size)
--> <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/
Tsunami/differentiate/NN_method_3.ipynb#ch0000001?line=115'>116</a> tf_sample_coords = tf.convert_to_tensor([tf
_coords[i] for i in indices])
    <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/
Tsunami/differentiate/NN_method_3.ipynb#ch0000001?line=116'>117</a> for k in range(n_trains):
    <a href='vscode-notebook-cell:/c%3A/Users/jtros/CS/cours/PoleProjet/FormationRecherche/Tsunami/TP/sceance4/
Tsunami/differentiate/NN_method_3.ipynb#ch0000001?line=117'>118</a>     if k % display_step == display_step-1:

File c:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami\venv_tsunami\lib\site-pa
ckages\tensorflow\python\util\traceback_utils.py:150, in filter_traceback.<locals>.error_handler(*args, **kwarg
s)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami\venv_tsunami\lib\site-pa
ckages\tensorflow\python\util\dispatch.py:1082, in add_dispatch_support.<locals>.decorator.<locals>.op_dispatch
_handler(*args, **kwargs)
   1080 # Fallback dispatch system (dispatch v1):
   1081 try:
-> 1082   return dispatch_target(*args, **kwargs)
   1083 except (TypeError, ValueError):
   1084   # Note: convert_to_eager_tensor currently raises a ValueError, not a
   1085   # TypeError, when given unexpected types.  So we need to catch both.
   1086   result = dispatch(op_dispatch_handler, args, kwargs)

File c:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami\venv_tsunami\lib\site-pa
ckages\tensorflow\python\ops\array_ops.py:1097, in _slice_helper(tensor, slice_spec, var)
   1095   var_empty = constant([], dtype=dtypes.int32)
   1096   packed_begin = packed_end = packed_strides = var_empty
-> 1097 return strided_slice(
   1098     tensor,
   1099     packed_begin,
   1100     packed_end,
   1101     packed_strides,
   1102     begin_mask=begin_mask,
   1103     end_mask=end_mask,
   1104     shrink_axis_mask=shrink_axis_mask,
   1105     new_axis_mask=new_axis_mask,
   1106     ellipsis_mask=ellipsis_mask,
   1107     var=var,
   1108     name=name)

File c:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami\venv_tsunami\lib\site-pa
ckages\tensorflow\python\util\traceback_utils.py:150, in filter_traceback.<locals>.error_handler(*args, **kwarg
s)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami\venv_tsunami\lib\site-pa
ckages\tensorflow\python\util\dispatch.py:1082, in add_dispatch_support.<locals>.decorator.<locals>.op_dispatch
_handler(*args, **kwargs)
   1080 # Fallback dispatch system (dispatch v1):
   1081 try:
-> 1082   return dispatch_target(*args, **kwargs)
   1083 except (TypeError, ValueError):
   1084   # Note: convert_to_eager_tensor currently raises a ValueError, not a
   1085   # TypeError, when given unexpected types.  So we need to catch both.
   1086   result = dispatch(op_dispatch_handler, args, kwargs)

File c:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami\venv_tsunami\lib\site-pa
ckages\tensorflow\python\ops\array_ops.py:1270, in strided_slice(input_, begin, end, strides, begin_mask, end_m
ask, ellipsis_mask, new_axis_mask, shrink_axis_mask, var, name)
   1267 if strides is None:
   1268   strides = ones_like(begin)
-> 1270 op = gen_array_ops.strided_slice(
   1271     input=input_,
   1272     begin=begin,
   1273     end=end,
   1274     strides=strides,
   1275     name=name,
   1276     begin_mask=begin_mask,
   1277     end_mask=end_mask,
   1278     ellipsis_mask=ellipsis_mask,
   1279     new_axis_mask=new_axis_mask,
   1280     shrink_axis_mask=shrink_axis_mask)
   1282 parent_name = name
   1284 if var is not None:

File c:\Users\jtros\CS\cours\PoleProjet\FormationRecherche\Tsunami\TP\sceance4\Tsunami\venv_tsunami\lib\site-pa
ckages\tensorflow\python\ops\gen_array_ops.py:10664, in strided_slice(input, begin, end, strides, begin_mask, e
nd_mask, ellipsis_mask, new_axis_mask, shrink_axis_mask, name)
  10662 if tld.is_eager:
  10663   try:
> 10664     _result = pywrap_tfe.TFE_Py_FastPathExecute(
  10665       _ctx, "StridedSlice", name, input, begin, end, strides, "begin_mask",
  10666       begin_mask, "end_mask", end_mask, "ellipsis_mask", ellipsis_mask,
  10667       "new_axis_mask", new_axis_mask, "shrink_axis_mask", shrink_axis_mask)
  10668     return _result
  10669   except _core._NotOkStatusException as e:

KeyboardInterrupt:
```