In [1]:
```python
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
import numpy as np
```

In [4]:
```python
__file__ = 'C:/Users/antle/Documents/Pole_recherche/Tsunami'
```

In [2]:
```python
__file__ = 'C:/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceance4/Tsunami'
```

In [3]:
```python
import os

print(os.getcwd())
os.chdir(__file__)
print(os.getcwd())
```

```
C:\Users\Gilles\CS\cours\PoleProjet\FormationRecherche\Tsunami\TF\sceance4\Tsunami\differentiate
C:\Users\Gilles\CS\cours\PoleProjet\FormationRecherche\Tsunami\TF\sceance4\Tsunami
```

In [4]:
```python
DTYPE = 'float32'
tf.keras.backend.set_floatx(DTYPE)
```

Create the model

In [85]:
```python
# To fix weights or biases of layers
class ConstantTensorInitializer(tf.keras.initializers.Initializer):
    """Initializes tensors to `t`."""

    def __init__(self, t):
        self.t = t

    def __call__(self, shape, dtype=None):
        return self.t

    def get_config(self):
        # A constraint config is a Python dictionary (JSON-serializable) that can be used to reinstantiate the same
        return {'t': self.t}

class ConstantTensorConstraint(tf.keras.constraints.Constraint):
    """Constrains tensors to `t`."""

    def __init__(self, t):
        self.t = t

    def __call__(self, w):
        return self.t

    def get_config(self):
        return {'t': self.t}


# t could be :
t = tf.ones((3,4))
print(t)

t1 = tf.eye(3,4)
print(t1)
```

```
tf.Tensor(
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]], shape=(3, 4), dtype=float32)
tf.Tensor(
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]], shape=(3, 4), dtype=float32)
```

In [5]:
```python
def generate_model(n_hidden=1,noise=False):
    # methods API Sequential
    multilayer_perceptron = keras.models.Sequential([
        keras.layers.Input(shape=(2))
    ])
    if noise:
        multilayer_perceptron.add(keras.layers.GaussianNoise(stddev=1e-3))
    for _ in range(n_hidden):
        multilayer_perceptron.add(keras.layers.Dense(20,activation='elu', kernel_initializer='he_normal'))
    multilayer_perceptron.add(keras.layers.Dense(1,use_bias=False))
    multilayer_perceptron.summary()
    return multilayer_perceptron
```

In [34]:
```python
# to check :
multilayer_perceptron = generate_model()
```

Model: "sequential_10"

Layer (type)	Output Shape	Param #
dense_30 (Dense)	(None, 20)	60
dense_31 (Dense)	(None, 20)	420
dense_32 (Dense)	(None, 1)	20

```
Total params: 500
Trainable params: 500
Non-trainable params: 0
```

In [6]:
```python
# 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


grid_length = 100


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.con

#print(boundary_coords.shape)
```

Method 1: mini-batch gradient descent with training to respect the boundary conditions at the same time of the one for the EDP

Papier : 2019_A_unified_deep_artificial_neural_network_approach_to_PDE_in_complex_geometries.pdf

In [11]:
```python
# method 1: mini-batch gradient descent + (loss = MSE + MSE on all boundary)

# a few train parameters to adjust
learning_rate = 0.01
training_steps = 200
batch_size = 1000
display_step = 100

alpha = 15 # magnify the boundary condition in the loss

optimizer = tf.optimizers.Adam(lr=learning_rate)

# generate model
multilayer_perceptron = generate_model()


# Universal Approximator
def g_1(x):
    # F_x = Fstud._eval_polynome_numpy(F_app_real,x[0,0],x[0,1])
    N_x = multilayer_perceptron(x)
    return N_x


# Custom loss function to approximate the derivatives

def custom_loss_1():
    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])

    _,dg_dxx,_,dg_dyy = differentiate(g_1,tf_sample_coords)
    f_r = tf.reshape(f(tf_sample_coords),[batch_size,1])
    res = residual(dg_dxx,dg_dyy,f_r)

    loss = tf.reduce_mean(tf.square(res)) + alpha*tf.reduce_mean(tf.square(g_1(tf_boundary_coords)-boundary_condi
    return loss

# train of method 1:
def train_step_1():
    with tf.GradientTape() as tape:
        loss = custom_loss_1()
    trainable_variables=multilayer_perceptron.trainable_variables
    gradients = tape.gradient(loss, trainable_variables)
    optimizer.apply_gradients(zip(gradients, trainable_variables))
    return loss
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 20)	60
dense_7 (Dense)	(None, 20)	420
dense_8 (Dense)	(None, 1)	20

```
Total params: 500
Trainable params: 500
Non-trainable params: 0
```

In [12]:
```python
# Training the Model of method 1:

all_losses = []

for i in range(training_steps):
    print('epoch:',i)
    loss = train_step_1()
    if i % display_step == display_step-1:
        print("loss:", loss)
    all_losses.append(loss)

plt.plot(np.arange(0,training_steps),all_losses)
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```

```
epoch: 0
epoch: 1
epoch: 2
epoch: 3
epoch: 4
epoch: 5
epoch: 6
epoch: 7
epoch: 8
epoch: 9
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
c:\Users\Gilles\CS\cours\PoleProjet\FormationRecherche\Tsunami\TF\sceance4\Tsunami\differentiate\NN_test_functi
on_A_F.ipynb Cell 14' in <cell line: 5>()
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000013?line=4'>5</a> for i in range(training_steps):
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000013?line=5'>6</a>     print('epoch:',i)
----> <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceance
4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000013?line=6'>7</a>     loss = train_step_1()
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000013?line=7'>8</a>     if i % display_step == display_st
ep-1:
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000013?line=8'>9</a>         print("loss:", loss)

c:\Users\Gilles\CS\cours\PoleProjet\FormationRecherche\Tsunami\TF\sceance4\Tsunami\differentiate\NN_test_functi
on_A_F.ipynb Cell 13' in train_step_1()
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=37'>38</a> def train_step_1():
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=38'>39</a>     with tf.GradientTape() as tape:
---> <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceance
4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=39'>40</a>         loss = custom_loss_1()
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=40'>41</a>     trainable_variables=multilayer_p
erceptron.trainable_variables
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=41'>42</a>     gradients = tape.gradient(loss,
 trainable_variables)

c:\Users\Gilles\CS\cours\PoleProjet\FormationRecherche\Tsunami\TF\sceance4\Tsunami\differentiate\NN_test_functi
on_A_F.ipynb Cell 13' in custom_loss_1()
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=25'>26</a> def custom_loss_1():
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=26'>27</a>     indices = np.random.randint(tf_c
oords.shape[0],size=batch_size)
---> <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceance
4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=27'>28</a>     tf_sample_coords = tf.convert_to
_tensor([tf_coords[i] for i in indices])
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=29'>30</a>     _,dg_dxx,_,dg_dyy = differentiat
e(g_1,tf_sample_coords)
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=30'>31</a>     f_r = tf.reshape(f(tf_sample_coo
rds),[batch_size,1])

c:\Users\Gilles\CS\cours\PoleProjet\FormationRecherche\Tsunami\TF\sceance4\Tsunami\differentiate\NN_test_functi
on_A_F.ipynb Cell 13' in <listcomp>(.0)
      ...
---> <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceance
4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000012?line=27'>28</a>     tf_sample_coords = tf.convert_to
_tensor([tf_coords[i] for i in indices])
      ...

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\util\traceback_
utils.py:150, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    <a href='file:///c%3A/Users/Gilles/AppData/Local/Programs/Python/Python39/lib/site-packages/tensorflow/pyth
on/util/traceback_utils.py?line=147'>148</a> filtered_tb = None
    <a href='file:///c%3A/Users/Gilles/AppData/Local/Programs/Python/Python39/lib/site-packages/tensorflow/pyth
on/util/traceback_utils.py?line=148'>149</a> try:
--> <a href='file:///c%3A/Users/Gilles/AppData/Local/Programs/Python/Python39/lib/site-packages/tensorflow/pytho
n/util/traceback_utils.py?line=149'>150</a>   return fn(*args, **kwargs)
    <a href='file:///c%3A/Users/Gilles/AppData/Local/Programs/Python/Python39/lib/site-packages/tensorflow/pyth
on/util/traceback_utils.py?line=150'>151</a> except Exception as e:
    <a href='file:///c%3A/Users/Gilles/AppData/Local/Programs/Python/Python39/lib/site-packages/tensorflow/pyth
on/util/traceback_utils.py?line=151'>152</a>   filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\util\dispatch.p
y:1082, in add_dispatch_support.<locals>.op_dispatch_handler(*args, **kwargs)
    ... # Fallback dispatch system (dispatch v1):
    ... try:
--> ...   return dispatch_target(*args, **kwargs)
    ... except (TypeError, ValueError):
    ...   # Note: convert_to_eager_tensor currently raises a ValueError, not a
    ...   # TypeError, when given unexpected types.  So we need to catch both.
    ...   result = dispatch(op_dispatch_handler, args, kwargs)

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\ops\array_ops.p
y:1052, in _slice_helper(tensor, slice_spec, var)
    ... var_empty = constant([], dtype=dtypes.int32)
    ... packed_begin = packed_end = packed_strides = var_empty
    ... try:
--> ...   return strided_slice(
    ...       tensor,
    ...       packed_begin,
    ...       packed_end,
    ...       packed_strides,
    ...       begin_mask=begin_mask,
    ...       end_mask=end_mask,
    ...       shrink_axis_mask=shrink_axis_mask,
    ...       new_axis_mask=new_axis_mask,
    ...       ellipsis_mask=ellipsis_mask,
    ...       var=var,
    ...       name=name)

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\util\traceback_
utils.py:150, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    ... filtered_tb = None
    ... try:
--> ...   return fn(*args, **kwargs)
    ... except Exception as e:
    ...   filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\util\dispatch.p
y:1082, in add_dispatch_support.<locals>.op_dispatch_handler(*args, **kwargs)
    ... # Fallback dispatch system (dispatch v1):
    ... try:
--> ...   return dispatch_target(*args, **kwargs)
    ... except (TypeError, ValueError):
    ...   # Note: convert_to_eager_tensor currently raises a ValueError, not a
    ...   # TypeError, when given unexpected types.  So we need to catch both.
    ...   result = dispatch(op_dispatch_handler, args, kwargs)

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\ops\array_ops.p
y:1225, in strided_slice(input_, begin, end, strides, begin_mask, end_mask, ellipsis_mask, new_axis_mask, shrin
k_axis_mask, var, name)
    ... if strides is None:
    ...   strides = ones_like(begin)
--> ... op = gen_array_ops.strided_slice(
    ...     input=input_,
    ...     begin=begin,
    ...     end=end,
    ...     strides=strides,
    ...     name=name,
    ...     begin_mask=begin_mask,
    ...     end_mask=end_mask,
    ...     ellipsis_mask=ellipsis_mask,
    ...     new_axis_mask=new_axis_mask,
    ...     shrink_axis_mask=shrink_axis_mask)
    ... parent_name = name
    ... if var is not None:

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\ops\gen_array_o
ps.py:10664, in strided_slice(input, begin, end, strides, begin_mask, end_mask, ellipsis_mask, new_axis_mask, s
hrink_axis_mask, name)
    ... if tld.is_eager:
    ...   try:
--> ...     _result = pywrap_tfe.TFE_Py_FastPathExecute(
    ...         _ctx, "StridedSlice", name, input, begin, end, strides, "beg
in_mask",
    ...         begin_mask, "end_mask", end_mask, "ellipsis_mask", ellipsis_
mask,
    ...         "new_axis_mask", new_axis_mask, "shrink_axis_mask", shrink_a
xis_mask)
    ...     return _result
    ...   except _core._NotOkStatusException as e:

KeyboardInterrupt:
```

Method 2: MBGD with pretraining to respect the boundary condition

In [14]:
```python
# method 2 : pretrain before on the boundary condition and after inside the boundary

# a few train parameters to adjust
learning_rate = 0.01
training_steps_boundary = 30
training_steps_global = 20
batch_size = 1000
display_step_boundary = 10
display_step_global = 10

alpha=4 # magnify the boundary condition in custom_loss_2()

optimizer = tf.optimizers.Adam(lr=learning_rate)

# Universal Approximator
def g_2(x):
    # F_x = Fstud._eval_polynome_numpy(F_app_real,x[0,0],x[0,1])
    N_x = multilayer_perceptron(x)
    return N_x

# generate model
multilayer_perceptron = generate_model()

# Custom loss function to approximate the derivatives
def custom_loss_2_boundary():
    indices = np.random.randint(tf_boundary_coords.shape[0],size=batch_size)
    tf_sample_boundary_coords = tf.convert_to_tensor([tf_boundary_coords[i] for i in indices])

    loss = tf.reduce_mean(tf.square(g_2(tf_sample_boundary_coords)-boundary_conditions(tf_sample_boundary_coords)
    # print('loss :',loss)
    return loss


def custom_loss_2():
    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])

    _,dg_dxx,_,dg_dyy = differentiate(g_2,tf_sample_coords)
    f_r = tf.reshape(f(tf_sample_coords),[batch_size,1])
    res = residual(dg_dxx,dg_dyy,f_r)

    loss = tf.reduce_mean(tf.square(res)) + alpha*tf.reduce_mean(tf.square(g_2(tf_boundary_coords)-boundary_condi
    return loss


# train of method 2:
def train_step_2_boundary():
    with tf.GradientTape() as tape:
        loss = custom_loss_2_boundary()
    trainable_variables=multilayer_perceptron.trainable_variables
    gradients = tape.gradient(loss, trainable_variables)
    optimizer.apply_gradients(zip(gradients, trainable_variables))
    return loss

def train_step_2_inside():
    with tf.GradientTape() as tape:
        loss = custom_loss_2()
    trainable_variables=multilayer_perceptron.trainable_variables
    gradients = tape.gradient(loss, trainable_variables)
    optimizer.apply_gradients(zip(gradients, trainable_variables))
    return loss
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 20)	60
dense_13 (Dense)	(None, 20)	420
dense_14 (Dense)	(None, 1)	20

```
Total params: 500
Trainable params: 500
Non-trainable params: 0
```

```
C:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:105: UserWa
rning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

In [15]:
```python
# Training the Model of method 2:

boundary_losses = []
b_global_losses = []
global_losses = []
for i in range(training_steps_boundary):
    print('epoch:',i)
    loss = train_step_2_boundary()
    if i % display_step_boundary == display_step_boundary-1:
        print("loss boundary: ", loss)
    boundary_losses.append(loss)
    b_global_losses.append(custom_loss_2())

for i in range(training_steps_global):
    print('epoch:',i)
    loss = train_step_2_inside()
    if i % display_step_global == display_step_global-1:
        print("loss :", loss)
    global_losses.append(loss)



plt.figure(1, figsize=(15, 7))

plt.subplot(211)
plt.plot(np.arange(0,training_steps_boundary),boundary_losses, color='blue', label='loss on the boundary')
plt.plot(np.arange(0,training_steps_boundary), b_global_losses, color="black", label='global loss during traini
plt.ylabel('loss')

plt.legend()

plt.subplot(212)
plt.plot(np.arange(0,training_steps_global+training_steps_boundary), b_global_losses+global_losses, color='red'
plt.ylabel('loss')
plt.legend()
plt.show()

print('final loss on the boundary:', custom_loss_2_boundary())
```

```
epoch: 0
epoch: 1
epoch: 2
epoch: 3
epoch: 4
epoch: 5
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
c:\Users\Gilles\CS\cours\PoleProjet\FormationRecherche\Tsunami\TF\sceance4\Tsunami\differentiate\NN_test_functi
on_A_F.ipynb Cell 17' in <cell line: 5>()
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000016?line=4'>5</a> for i in range(training_steps_bound
ary):
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000016?line=5'>6</a>     print('epoch:',i)
----> <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceance
4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000016?line=6'>7</a>     loss = train_step_2_boundary()
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000016?line=7'>8</a>     if i % display_step_boundary == d
isplay_step_boundary-1:
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000016?line=8'>9</a>         print("loss boundary: ", loss)

c:\Users\Gilles\CS\cours\PoleProjet\FormationRecherche\Tsunami\TF\sceance4\Tsunami\differentiate\NN_test_functi
on_A_F.ipynb Cell 16' in train_step_2_boundary()
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=46'>47</a> def train_step_2_boundary():
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=47'>48</a>     with tf.GradientTape() as tape:
---> <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceance
4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=48'>49</a>         loss = custom_loss_2_boundary
()
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=49'>50</a>     trainable_variables=multilayer_p
erceptron.trainable_variables
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=50'>51</a>     gradients = tape.gradient(loss,
 trainable_variables)

c:\Users\Gilles\CS\cours\PoleProjet\FormationRecherche\Tsunami\TF\sceance4\Tsunami\differentiate\NN_test_functi
on_A_F.ipynb Cell 16' in custom_loss_2_boundary()
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=24'>25</a> def custom_loss_2_boundary():
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=25'>26</a>     indices = np.random.randint(tf_b
oundary_coords.shape[0],size=batch_size)
---> <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceance
4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=26'>27</a>     tf_sample_boundary_coords = tf.c
onvert_to_tensor([tf_boundary_coords[i] for i in indices])
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=28'>29</a>     loss = tf.reduce_mean(tf.square
(g_2(tf_sample_boundary_coords)-boundary_conditions(tf_sample_boundary_coords)))
      <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceanc
e4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=29'>30</a>     # print('loss :',loss)

c:\Users\Gilles\CS\cours\PoleProjet\FormationRecherche\Tsunami\TF\sceance4\Tsunami\differentiate\NN_test_functi
on_A_F.ipynb Cell 16' in <listcomp>(.0)
      ...
---> <a href='vscode-notebook-cell:/c%3A/Users/Gilles/CS/cours/PoleProjet/FormationRecherche/Tsunami/TF/sceance
4/Tsunami/differentiate/NN_test_function_A_F.ipynb#ch0000015?line=26'>27</a>     tf_sample_boundary_coords = tf.c
onvert_to_tensor([tf_boundary_coords[i] for i in indices])
      ...

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\util\traceback_
utils.py:150, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    ... filtered_tb = None
    ... try:
--> ...   return fn(*args, **kwargs)
    ... except Exception as e:
    ...   filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\util\dispatch.p
y:1082, in add_dispatch_support.<locals>.op_dispatch_handler(*args, **kwargs)
    ... # Fallback dispatch system (dispatch v1):
    ... try:
--> ...   return dispatch_target(*args, **kwargs)
    ... except (TypeError, ValueError):
    ...   # Note: convert_to_eager_tensor currently raises a ValueError, not a
    ...   # TypeError, when given unexpected types.  So we need to catch both.
    ...   result = dispatch(op_dispatch_handler, args, kwargs)

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\ops\array_ops.p
y:1052, in _slice_helper(tensor, slice_spec, var)
    ... var_empty = constant([], dtype=dtypes.int32)
    ... packed_begin = packed_end = packed_strides = var_empty
    ... try:
--> ...   return strided_slice(
    ...       tensor,
    ...       packed_begin,
    ...       packed_end,
    ...       packed_strides,
    ...       begin_mask=begin_mask,
    ...       end_mask=end_mask,
    ...       shrink_axis_mask=shrink_axis_mask,
    ...       new_axis_mask=new_axis_mask,
    ...       ellipsis_mask=ellipsis_mask,
    ...       var=var,
    ...       name=name)

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\util\traceback_
utils.py:150, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    ... filtered_tb = None
    ... try:
--> ...   return fn(*args, **kwargs)
    ... except Exception as e:
    ...   filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\util\dispatch.p
y:1082, in add_dispatch_support.<locals>.op_dispatch_handler(*args, **kwargs)
    ... # Fallback dispatch system (dispatch v1):
    ... try:
--> ...   return dispatch_target(*args, **kwargs)
    ... except (TypeError, ValueError):
    ...   # Note: convert_to_eager_tensor currently raises a ValueError, not a
    ...   # TypeError, when given unexpected types.  So we need to catch both.
    ...   result = dispatch(op_dispatch_handler, args, kwargs)

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\ops\array_ops.p
y:1225, in strided_slice(input_, begin, end, strides, begin_mask, end_mask, ellipsis_mask, new_axis_mask, shrin
k_axis_mask, var, name)
    ... if strides is None:
    ...   strides = ones_like(begin)
--> ... op = gen_array_ops.strided_slice(
    ...     input=input_,
    ...     begin=begin,
    ...     end=end,
    ...     strides=strides,
    ...     name=name,
    ...     begin_mask=begin_mask,
    ...     end_mask=end_mask,
    ...     ellipsis_mask=ellipsis_mask,
    ...     new_axis_mask=new_axis_mask,
    ...     shrink_axis_mask=shrink_axis_mask)
    ... parent_name = name
    ... if var is not None:

File c:\Users\Gilles\AppData\Local\Programs\Python\Python39\lib\site-packages\tensorflow\python\ops\gen_array_o
ps.py:10664, in strided_slice(input, begin, end, strides, begin_mask, end_mask, ellipsis_mask, new_axis_mask, s
hrink_axis_mask, name)
    ... if tld.is_eager:
    ...   try:
--> ...     _result = pywrap_tfe.TFE_Py_FastPathExecute(
    ...         _ctx, "StridedSlice", name, input, begin, end, strides, "beg
in_mask",
    ...         begin_mask, "end_mask", end_mask, "ellipsis_mask", ellipsis_
mask,
    ...         "new_axis_mask", new_axis_mask, "shrink_axis_mask", shrink_a
xis_mask)
    ...     return _result
    ...   except _core._NotOkStatusException as e:

KeyboardInterrupt:
```

Method 3: g automatically respects the boundary conditions

article : 1997_Artificial_neural_networks_for_solving_ordinary_and_partial_differential_equations.pdf

What is the fastest way to differentiate a sympy expr ?

- see dynamic_diff.ipynb

diff_gradually is better. Should be kept to differentiate F and A.

--- done

Other librairies than sympy? autograd,...

TODO: compare different methods according to speed of differentiation

--- done in dynamic_diff.ipynb

We need to compute : $\Delta g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$

Since we set $g(X) := A(X) + F(X) \times N(X)$ with $F_{|\partial\Omega} = 0$ and $A_{|\partial\Omega} = 0$ to respect the homogeneous Dirichlet boundary condition, and N estimated by the model.

We need to know :

$$\frac{\partial^2 g}{\partial x^2} = \frac{\partial^2 F}{\partial x^2} N + 2\frac{\partial F}{\partial x}\frac{\partial N}{\partial x} + F\frac{\partial^2 N}{\partial x^2} + \frac{\partial^2 A}{\partial x^2}$$

and the same goes for the variable y

In [8]:
```python
# Set F here

import sympy as sm
from sympy import Matrix
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)
dexpr_F_dxx = sm.diff(dexpr_F_dx,x)
dexpr_F_dy = sm.diff(expr_F,y)
dexpr_F_dyy = sm.diff(dexpr_F_dy,y)

# print(dexpr_F_dx)
# print(dexpr_F_dxx)

# 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])))
    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
```

Dummy F

F of F_functions