Laboratorium 10

Sebastian Soczawa, Piotr Kuchta

Zadanie 1

Cząsteczka w dwuwymiarowej studni potencjału. Cząsteczka odbija się od ścian dwuwymiarowej nieskończonej studni potencjału o szerokości L. Zachowanie cząsteczki opisane jest bezczasowym równaniem Schr ̈odingera. Rozwiąż powyższe zagadnienie brzegowe (1),(2) dla (n1, n2) $\in \{1, 2\} \times \{1, 2\} = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$. Do rozwiązania użyj sieci neuronowych PINN (ang. Physics-informed Neural Network), wykorzystując bibliotekę DeepXDE. Warstwa wejściowa sieci powinna posiadać 4 neurony, L0 = (x, y, n1, n2), kodujące od- powiednio położenie cząstki (x, y) oraz liczby kwantowe n1, n2. Jako funkcję aktywacji przyjmij tangens hiperboliczny, tanh.

Importujemy niezbędne biblioteki. Jako backend będziemy używać biblioteki pytorch.

In [ ]:
```
import numpy as np
import deepxde as dde
import torch
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from functools import partial
from math import sin
```

Ustawiamy globalne własności.

In [ ]:
```
save_folder = "./visualization/"
dde.config.set_default_float("float64")
origin_ns = np.array([[1, 1], [1, 2], [2, 1], [2, 2]])
```

```
Set the default float type to float64
```

Definiujemy funkcje zwracjające dokładne rozwiązanie.

In [ ]:
```
def exact_sol(f,n1,n2):
    x = f[:, 0:1]
    y = f[:, 1:2]
    return np.sin(n1*np.pi*(x + 1)*0.5)*np.sin(n2*np.pi*(y + 1)*0.5)

def exact2(f,n1,n2):
    arr = []
    for x,y in f:
        tmp = sin(n1*np.pi*(x + 1)*0.5)*sin(n2*np.pi*(y + 1)*0.5)
        arr.append(tmp)
    return np.array(arr)
```

Definiujemy pde potrzebne stworzenia modeli rozwiązujących nasz problem.

In [ ]:
```
def pde(netw_in, netw_out, n1, n2):
    dy_xx = dde.grad.hessian(netw_out, netw_in, i=0, j=0)
    dy_yy = dde.grad.hessian(netw_out, netw_in, i=1, j=1)
    return 0.5*(dy_xx + dy_yy)+(n1**2+n2**2)*(np.pi**2)*netw_out/8

def boundary(_, on_boundary):
    return on_boundary
```

Definiujemy funkcje niezbędne do generowania wykresów.

In [ ]:
```
def generate_test_data(size):
    x = np.linspace(-1, 1, size)
    y = np.linspace(-1, 1, size)
    test_x, test_y = np.meshgrid(x, y)
    test_x = test_x.reshape(-1)
    test_y = test_y.reshape(-1)
    return np.vstack((test_x, test_y)).T


def get_solution(data,model):
    solution = np.hstack([model.predict(el) for el in data])
    return solution

def graph_shreodinger(x,y,z,title,fig,ax,i,j):
    fig.set_size_inches(10, 10)
    fig.suptitle(title)
    contour = ax[i][j].tricontour(x, y, z)
    ax[i][j].clabel(contour, inline=1, fontsize=10)

def start_values(points, values):
    init = []
    for i in range(len(points)):
        p = dde.PointSetBC(points[i], values[i])
        init.append(p)
    return init

def fun_value(ptr,n1,n2):
    value = np.sin(n1 * np.pi * (ptr[:, :, 0] + 1) / 2) * np.sin(n2 * np.
    return value.reshape(1, -1)[0]
```

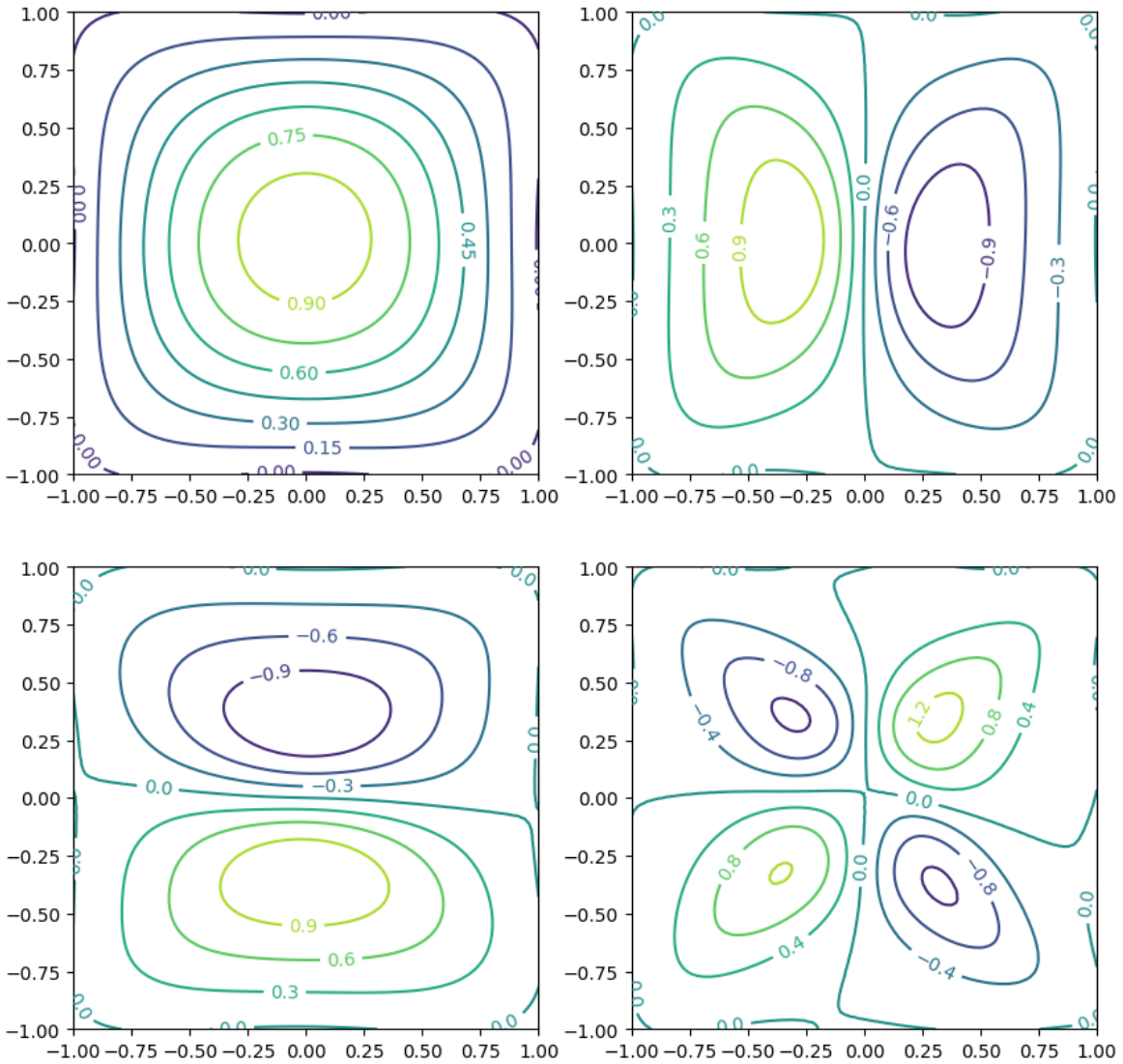
Tworzymy cztery modele dla każdej pary n1,n2. Zdecydowaliśmy się na takie rozwiązanie ponieważ wartiant jednego modelu z czterema neuronami wejściowymi dostarczał niepoprawne wyniki.

In [ ]:
```
geom = dde.geometry.Rectangle([-1, -1], [1, 1])
bc = dde.icbc.DirichletBC(geom, lambda x: 0, lambda x, on_boundary: on_bo
fig_p, ax_p = plt.subplots(nrows=2, ncols=2)
fig_p.set_size_inches(10, 10)
fig_e, ax_e = plt.subplots(nrows=2, ncols=2)
fig_e.set_size_inches(10, 10)
fig_r, ax_r = plt.subplots(nrows=2, ncols=2)
fig_r.set_size_inches(10, 10)
for i, ns in enumerate(origin_ns):
    conditions = [bc]
    new_pde = partial(pde,n1=ns[0],n2=[ns[1]])
    certain = np.array([[[-0.5, 0.5]], [[0.5, -0.5]],[[-0.5, -0.5]],[[0.5
    values = fun_value(certain, ns[0],ns[1])
    constr = start_values(certain, values)
    conditions.extend(constr)
    exn = partial(exact_sol,n1=ns[0],n2=ns[1])
    data = dde.data.PDE(
        geom,
        new_pde,
        conditions,
        num_domain = 800,
        num_boundary = 100,
        num_test=10**3,
        solution=exn,
    )
    layer_size = [2] + [20] * 3 + [1]
    activation = "tanh"
    initializer = "Glorot normal"

    net = dde.maps.FNN(layer_size, activation, initializer)


    model = dde.Model(data, net)
    model.compile("adam", lr=0.001, metrics=["l2 relative error"])

    losshistory, train_state = model.train(iterations=2000)
    test_domain = generate_test_data(100)
    predicted_solution = get_solution(test_domain,model)
    graph_shreodinger(test_domain[:,0],test_domain[:,1],predicted_solutio
    exact_solution = exact2(test_domain,ns[0],ns[1])
    graph_shreodinger(test_domain[:,0],test_domain[:,1],exact_solution,"e
    error_l2 = np.abs((predicted_solution - exact_solution)**2)

    graph_shreodinger(test_domain[:,0],test_domain[:,1],error_l2,"error",
```

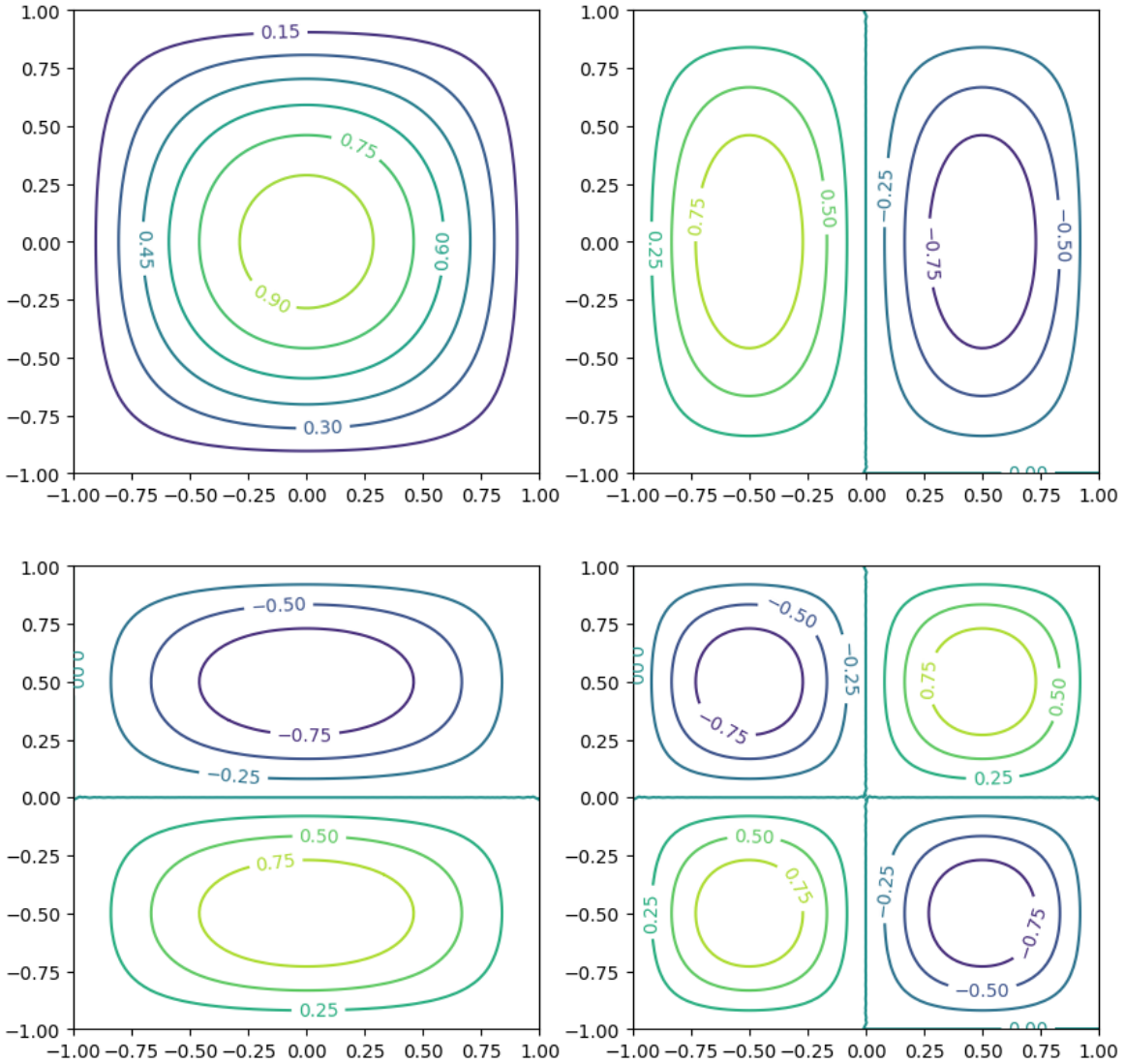
```
Warning: 1000 points required, but 1024 points sampled.
Compiling model...
'compile' took 0.000285 s

Training model...

Step      Train loss
Test loss                                                          Test metri
c
0         [1.74e-01, 3.61e-01, 1.59e-01, 1.17e-02, 7.95e-01, 1.00e+00]
[1.74e-01, 3.61e-01, 1.59e-01, 1.17e-02, 7.95e-01, 1.00e+00]    [1.19e+00]
1000      [5.36e-04, 2.08e-05, 2.72e-05, 2.11e-05, 2.67e-05, 7.29e-05]
[5.36e-04, 2.08e-05, 2.72e-05, 2.11e-05, 2.67e-05, 7.29e-05]    [2.80e-02]
2000      [1.31e-04, 6.47e-09, 1.38e-08, 1.43e-08, 3.16e-08, 1.60e-08]
[1.31e-04, 6.47e-09, 1.38e-08, 1.43e-08, 3.16e-08, 1.60e-08]    [2.90e-02]

Best model at step 2000:
  train loss: 1.31e-04
  test loss: 1.31e-04
  test metric: [2.90e-02]

'train' took 18.125513 s

Warning: 1000 points required, but 1024 points sampled.
Compiling model...
'compile' took 0.000355 s

Training model...

Step      Train loss
Test loss                                                          Test metri
c
0         [2.11e-01, 2.23e-01, 2.23e-01, 8.75e-02, 8.75e-02, 1.50e-32]
[2.11e-01, 2.23e-01, 2.23e-01, 8.75e-02, 8.75e-02, 1.50e-32]    [7.02e-01]
1000      [4.16e-04, 8.65e-07, 3.07e-07, 3.53e-06, 5.03e-06, 4.66e-08]
[4.16e-04, 8.65e-07, 3.07e-07, 3.53e-06, 5.03e-06, 4.66e-08]    [2.15e-01]
2000      [2.82e-04, 2.73e-05, 3.37e-06, 1.91e-05, 5.36e-05, 5.47e-06]
[2.82e-04, 2.73e-05, 3.37e-06, 1.91e-05, 5.36e-05, 5.47e-06]    [2.36e-01]

Best model at step 2000:
  train loss: 3.91e-04
  test loss: 3.91e-04
  test metric: [2.36e-01]

'train' took 17.658477 s

Warning: 1000 points required, but 1024 points sampled.
Compiling model...
'compile' took 0.000225 s

Training model...

Step      Train loss
Test loss                                                          Test metri
c
0         [1.01e-01, 6.11e-01, 6.11e-01, 9.97e-01, 9.97e-01, 1.50e-32]
[1.01e-01, 6.11e-01, 6.11e-01, 9.97e-01, 9.97e-01, 1.50e-32]    [1.35e+00]
1000      [1.34e-03, 2.46e-06, 2.96e-06, 7.60e-08, 1.47e-09, 2.76e-12]
[1.34e-03, 2.46e-06, 2.96e-06, 7.60e-08, 1.47e-09, 2.76e-12]    [2.22e-01]
2000      [2.57e-04, 3.58e-08, 2.87e-08, 6.08e-09, 4.85e-08, 2.10e-11]
```

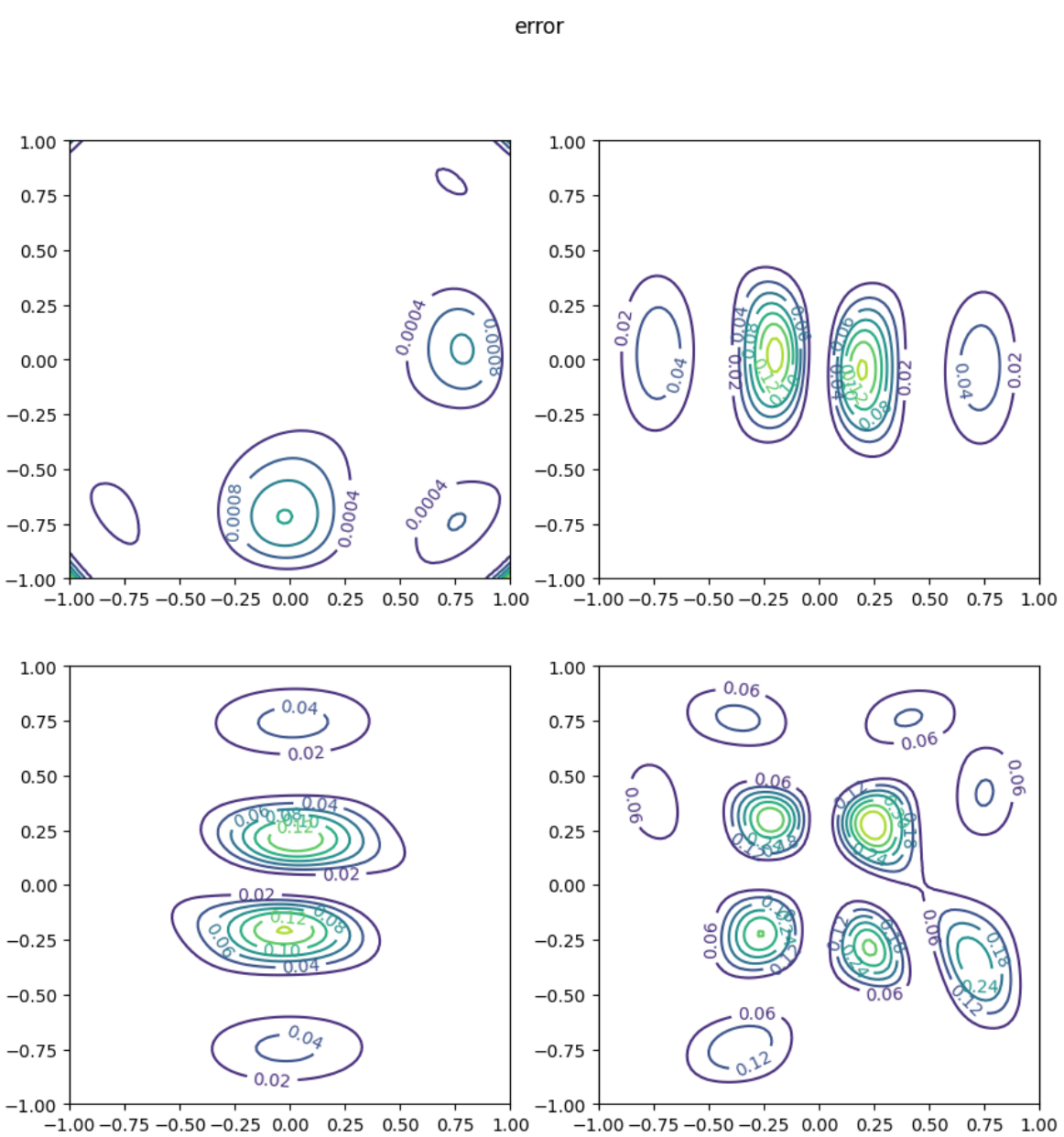
```
[2.57e-04, 3.58e-08, 2.87e-08, 6.08e-09, 4.85e-08, 2.10e-11]    [2.31e-01]

Best model at step 2000:
  train loss: 2.57e-04
  test loss: 2.57e-04
  test metric: [2.31e-01]

'train' took 17.803796 s

Warning: 1000 points required, but 1024 points sampled.
Compiling model...
'compile' took 0.000253 s

Training model...

Step      Train loss
Test loss                                                    Test metri
c
0         [1.56e-01, 1.33e+00, 7.20e-01, 2.05e+00, 3.22e-01, 2.25e-64]
[1.56e-01, 1.33e+00, 7.20e-01, 2.05e+00, 3.22e-01, 2.25e-64]    [1.21e+00]
1000      [4.81e-04, 8.78e-08, 1.46e-07, 4.96e-09, 5.78e-08, 1.86e-08]
[4.81e-04, 8.78e-08, 1.46e-07, 4.96e-09, 5.78e-08, 1.86e-08]    [4.72e-01]
2000      [9.10e-05, 1.16e-09, 8.77e-09, 1.23e-08, 1.97e-09, 4.15e-10]
[9.10e-05, 1.16e-09, 8.77e-09, 1.23e-08, 1.97e-09, 4.15e-10]    [4.65e-01]

Best model at step 2000:
  train loss: 9.10e-05
  test loss: 9.10e-05
  test metric: [4.65e-01]

'train' took 17.972253 s
```

predicted


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
0.75
0.90
0.60
0.45
0.30
0.15
0.00
0.3
0.6
0.9
0.0
−0.3
−0.6
−0.9
−0.6
−0.9
0.0
−0.3
0.9
0.6
0.3
−0.8
−0.4
1.2
0.8
0.4
0.0
0.8
0.4
−0.8
−0.4

exact


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
0.15
0.30
0.45
0.60
0.75
0.90
0.25
0.50
0.75
−0.25
−0.50
−0.75
0.00
−0.50
−0.75
−0.25
0.00
0.50
0.75
0.25
0.50
0.75
0.25
−0.25
−0.75
−0.50



Wnioski

Wyniki przewidywane przez modele dość znacząco różnią się od rzeczywistych. Może być to spowodowane zbyt małą ilością punktów wewnątrz badanego obszaru bądź małą liczbą iteracji.