project

May 22, 2019

1

In [2]:
```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
from sklearn.metrics import mean_squared_error
import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
global pi
pi = np.pi
```

In [4]:
```
xmin, xmax = 0.0, 1.0
ymin, ymax = 0.0, 1.0
def phi_1(x, y):
    return np.sin(2.0*pi*x)**2*np.cos(4.0*pi*y) + np.sin(4.0*pi*x)*np.cos(2.0*pi*y)**2
def phi_2(x, y):
    return y*(1 - y)*x**3
def phi_3(x, y):
    return (1.0 - x**2)*(2.0*y**3 - 3.0*y**2 + 1.0)
```

In [5]:
```
def func_1(x, y):
    return 8.0*pi**2*np.cos(4.0*pi*y)*(np.cos(4.0*pi*x) - np.sin(4.0*pi*x)) - \
           16.0*pi**2*(np.sin(4.0*pi*x)*np.cos(2.0*pi*y)**2 + np.sin(2.0*pi*x)**2 * np
def func_2(x, y):
    return 6.0*x*y*(1.0 - y) - 2.0*x**3
def func_3(x, y):
    return -2.0*(2.0*y**3 - 3.0*y**2 + 1.0) + 6.0*(1.0 - x**2)*(2.0*y - 1.0)
```

In [6]:
```
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
def plot_loss(loss, N, func):
```

```
            plt.figure(figsize=(13, 7))
            plt.xlabel('  ', fontsize='xx-large')
            plt.ylabel('MSE error', fontsize='xx-large')
            plt.plot(N, loss, '.-', color='g')
            plt.grid()
            plt.savefig('results/fft_loss_' + func.__name__+ '.png', bbox_inches='tight')
            plt.show();
        def plot_fft(solution, func):
            plt.figure(figsize=(10, 10))
            plt.imshow(np.transpose(solution), extent=[xmin, xmax, ymin, ymax])
            plt.title(func.__name__ + str(solution.shape), fontsize=30)
            plt.xlabel('x', fontsize=40)
            plt.ylabel('y', fontsize=40)
            plt.colorbar()
            plt.tight_layout()
            plt.savefig('results/fft_' + func.__name__+ str(solution.shape) + '.png', bbox_incl
```

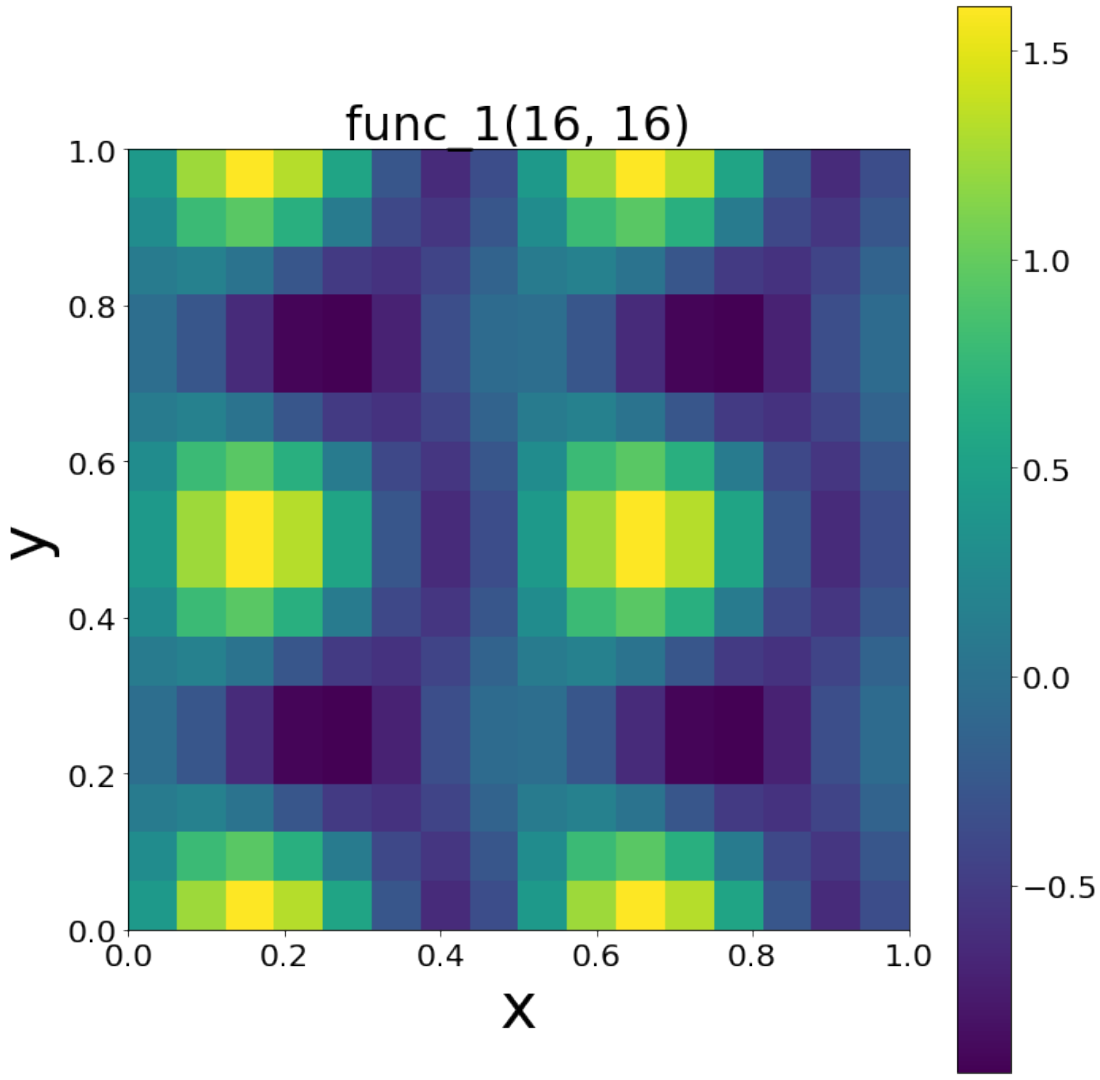
In [7]:
```
def fft(Nx, Ny, func, phi):
    dx = (xmax - xmin)/Nx
    dy = (ymax - ymin)/Ny
    x = (np.arange(Nx) + 0.5)*dx
    y = (np.arange(Ny) + 0.5)*dy
    x2d = np.repeat(x, Ny)
    x2d.shape = (Nx, Ny)
    y2d = np.repeat(y, Nx)
    y2d.shape = (Ny, Nx)
    y2d = np.transpose(y2d)
    f = func(x2d, y2d)
    F = np.fft.fft2(f)
    kx = np.fft.fftfreq(Nx)/dx
    ky = np.fft.fftfreq(Ny)/dy
    kx2d = np.repeat(kx, Ny)
    kx2d.shape = (Nx, Ny)
    ky2d = np.repeat(ky, Nx)
    ky2d.shape = (Ny, Nx)
    ky2d = np.transpose(ky2d)
    zero_singularity = F[0,0]
    F = 0.5*F / ((np.cos(2.0*pi*kx2d/Nx) - 1.0)/dx**2 + (np.cos(2.0*pi*ky2d/Ny) - 1.0),
    F[0,0] = zero_singularity
    solution = np.real(np.fft.ifft2(F))
    loss = mean_squared_error(solution, phi(x2d,y2d))
    return loss, solution
```

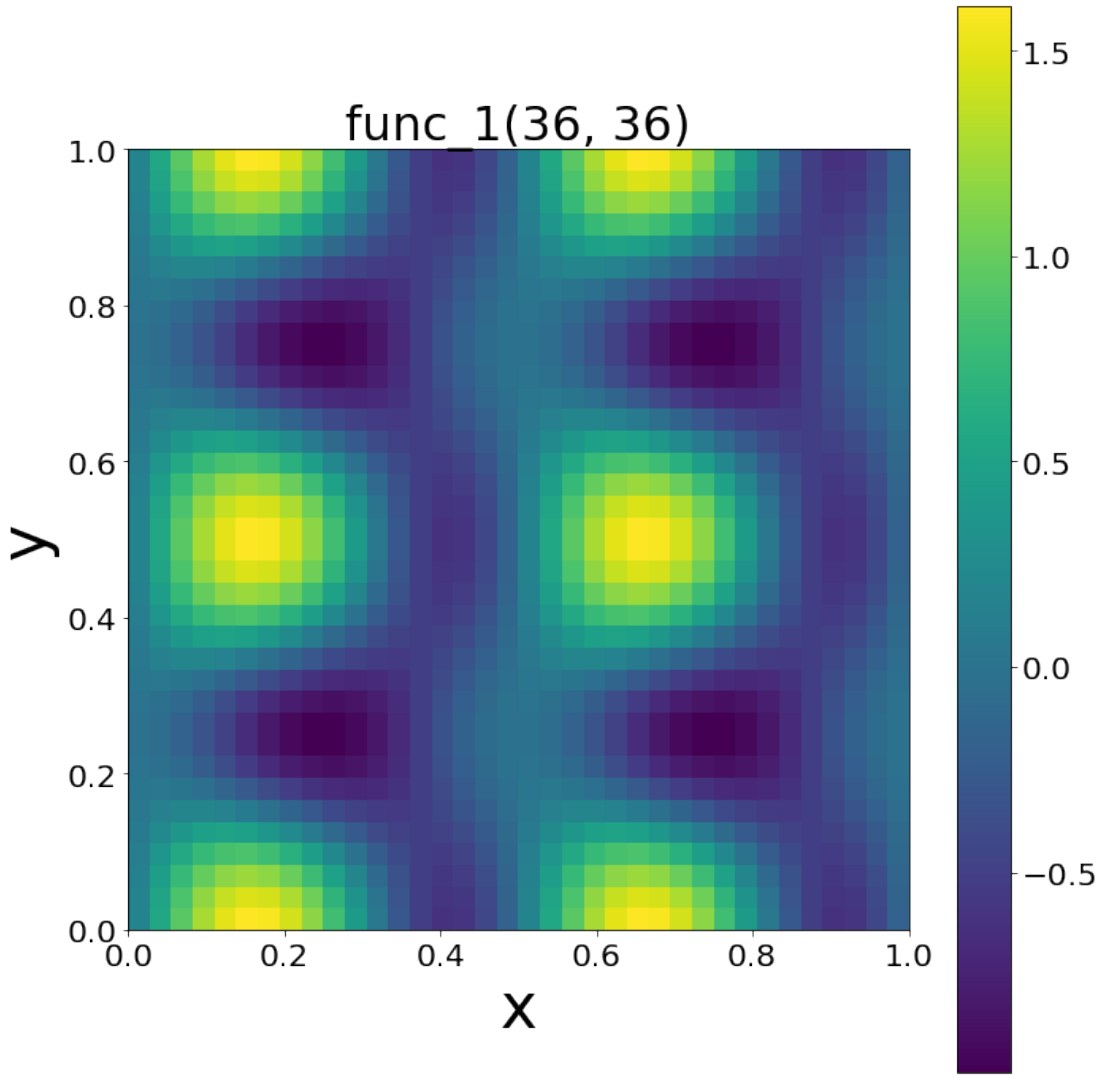
In [8]:
```
N = [10, 16, 20, 25, 32, 36, 40, 45]
N.extend([i for i in range(50, 500, 40)])
def get_results(N, loss, func, phi):
    for n in N:
        loss.append(fft(n, n, func, phi)[0])
```

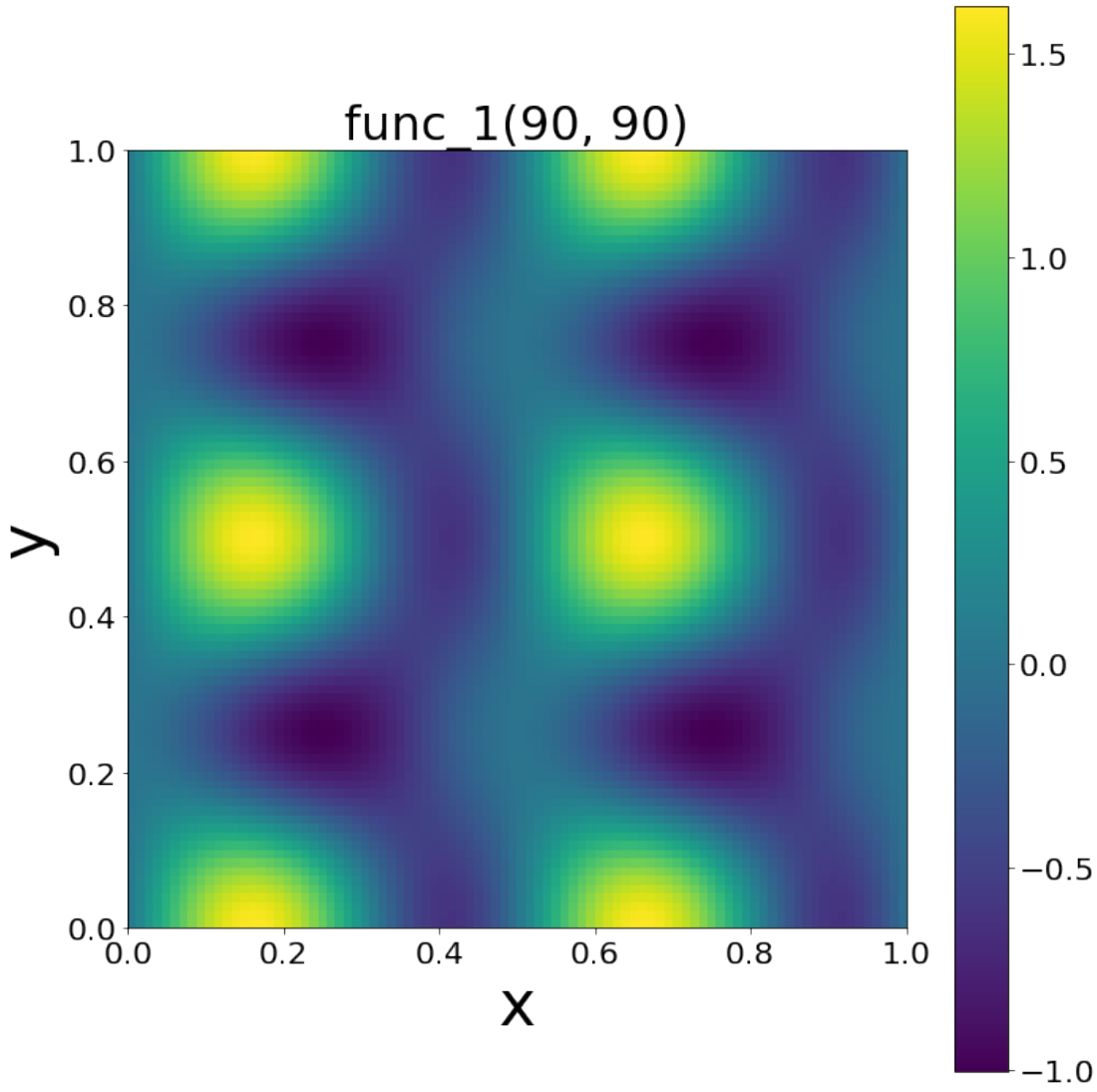
```
            solution = fft(n, n, func, phi)[1]
            if n == 16 or n == 36 or n == 90 or n == 450:
                plot_fft(solution, func)
        print(' :\n{}'.format(N))
        print('     :\n{}'.format(np.array(loss, dtype=np.float64)))
        #plot_loss(loss, N)
        return solution, loss
```

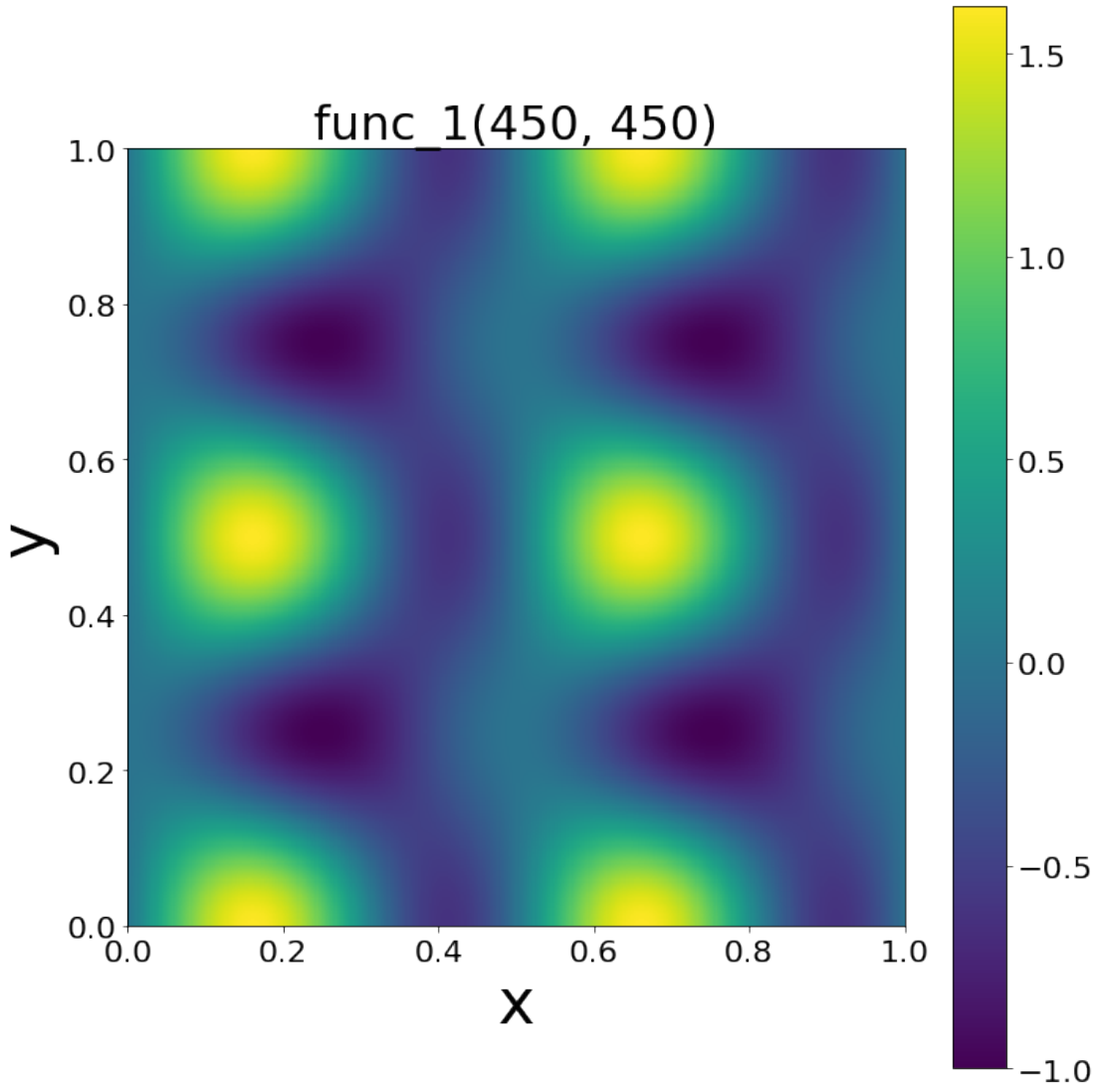
In [10]:
```
loss_1 = []
solution_1, loss_1 = get_results(N, loss_1, func_1, phi_1)
```

```
 :
[10, 16, 20, 25, 32, 36, 40, 45, 50, 90, 130, 170, 210, 250, 290, 330, 370, 410, 450, 490]
     :
[7.63345710e-03 1.05453950e-03 4.22312105e-04 1.70514783e-04
 6.28956902e-05 3.91380750e-05 2.56189191e-05 1.59605840e-05
 1.04562029e-05 9.91710967e-07 2.27583453e-07 7.77946748e-08
 3.34030846e-08 1.66286859e-08 9.18327922e-09 5.47665584e-09
 3.46538452e-09 2.29833645e-09 1.58376975e-09 1.12655519e-09]
```

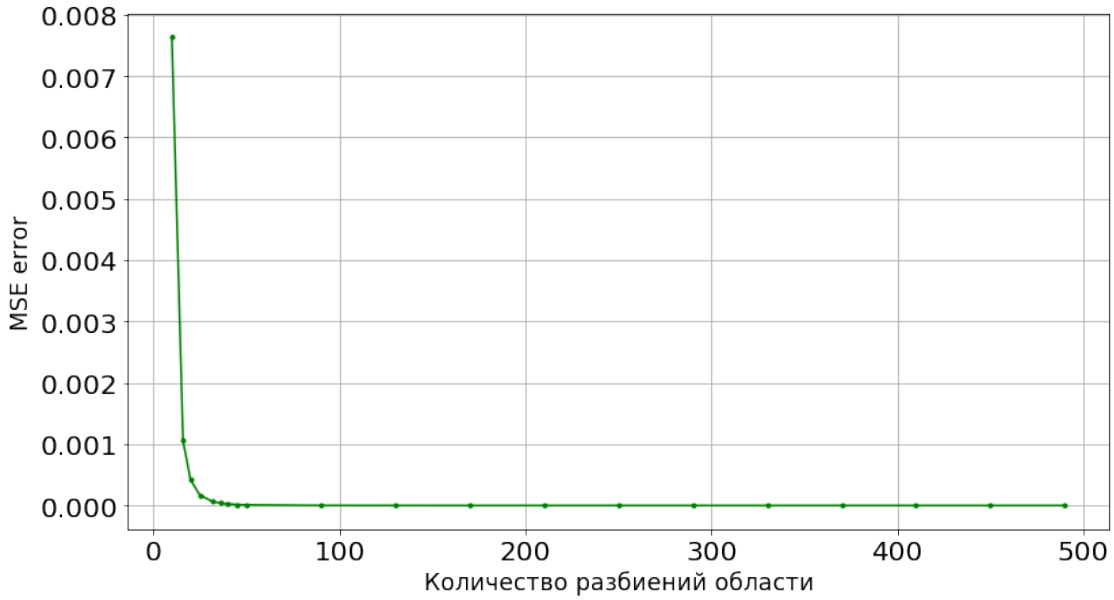

func_1(16, 16)
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
1.5
1.0
0.5
0.0
−0.5


func_1(36, 36)
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
1.5
1.0
0.5
0.0
−0.5

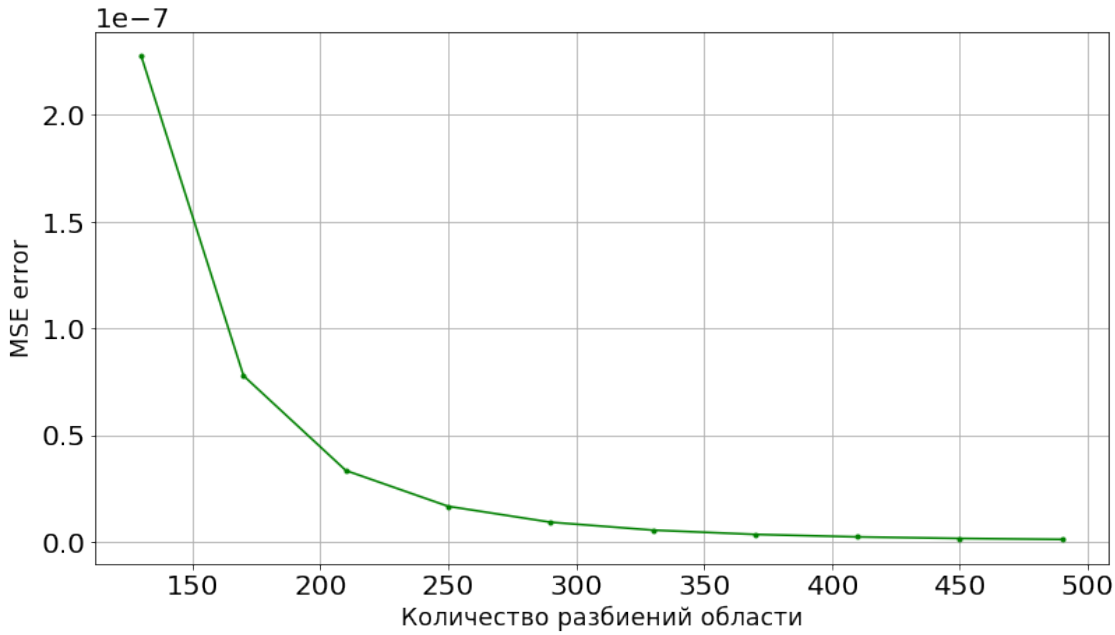

func_1(90, 90)
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
1.5
1.0
0.5
0.0
−0.5
−1.0



In [12]: `plot_loss(loss_1, N, func_1)`



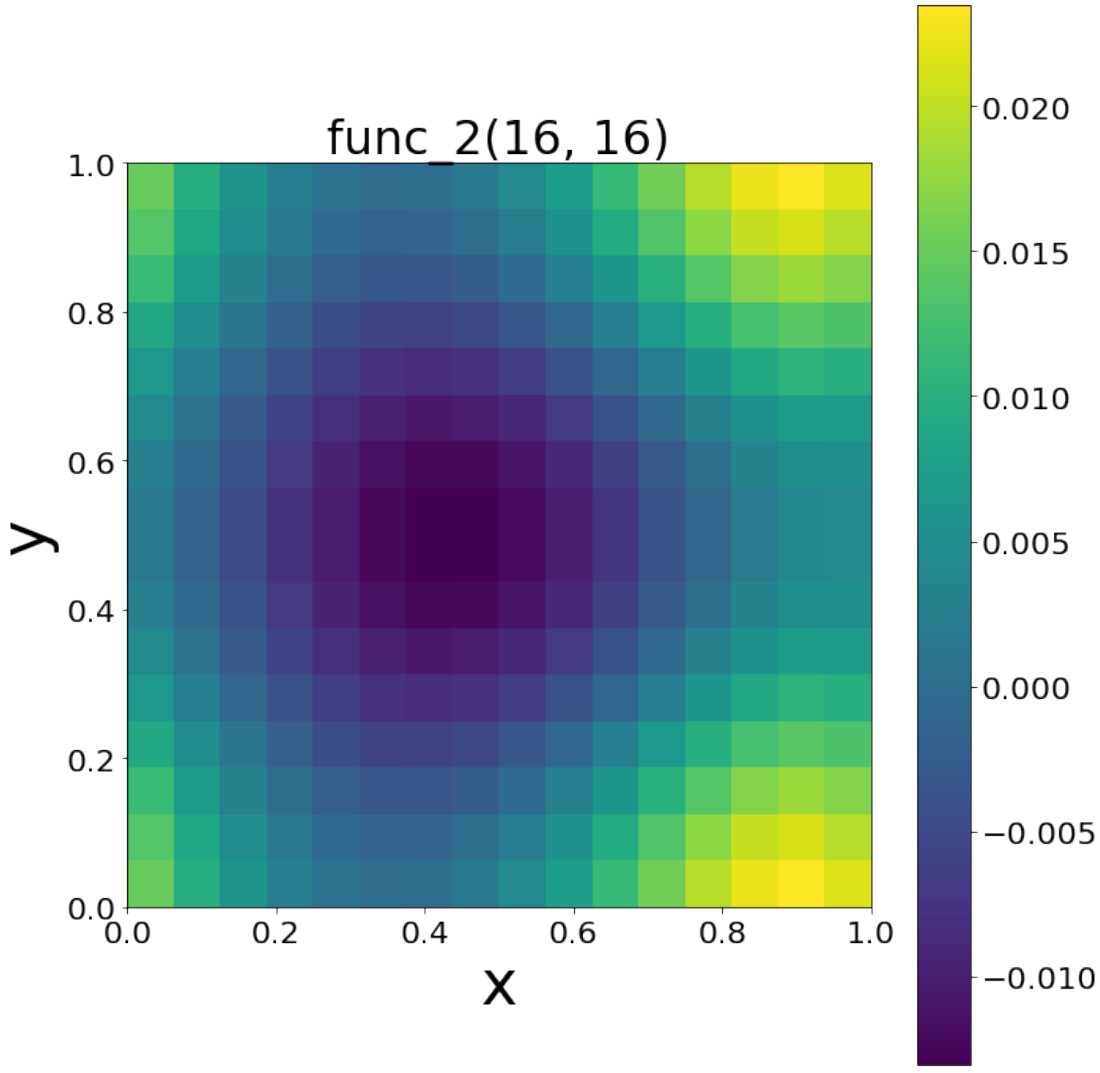
In [11]:
```
plot_loss(loss_1[10:], N[10:], func_1)
```

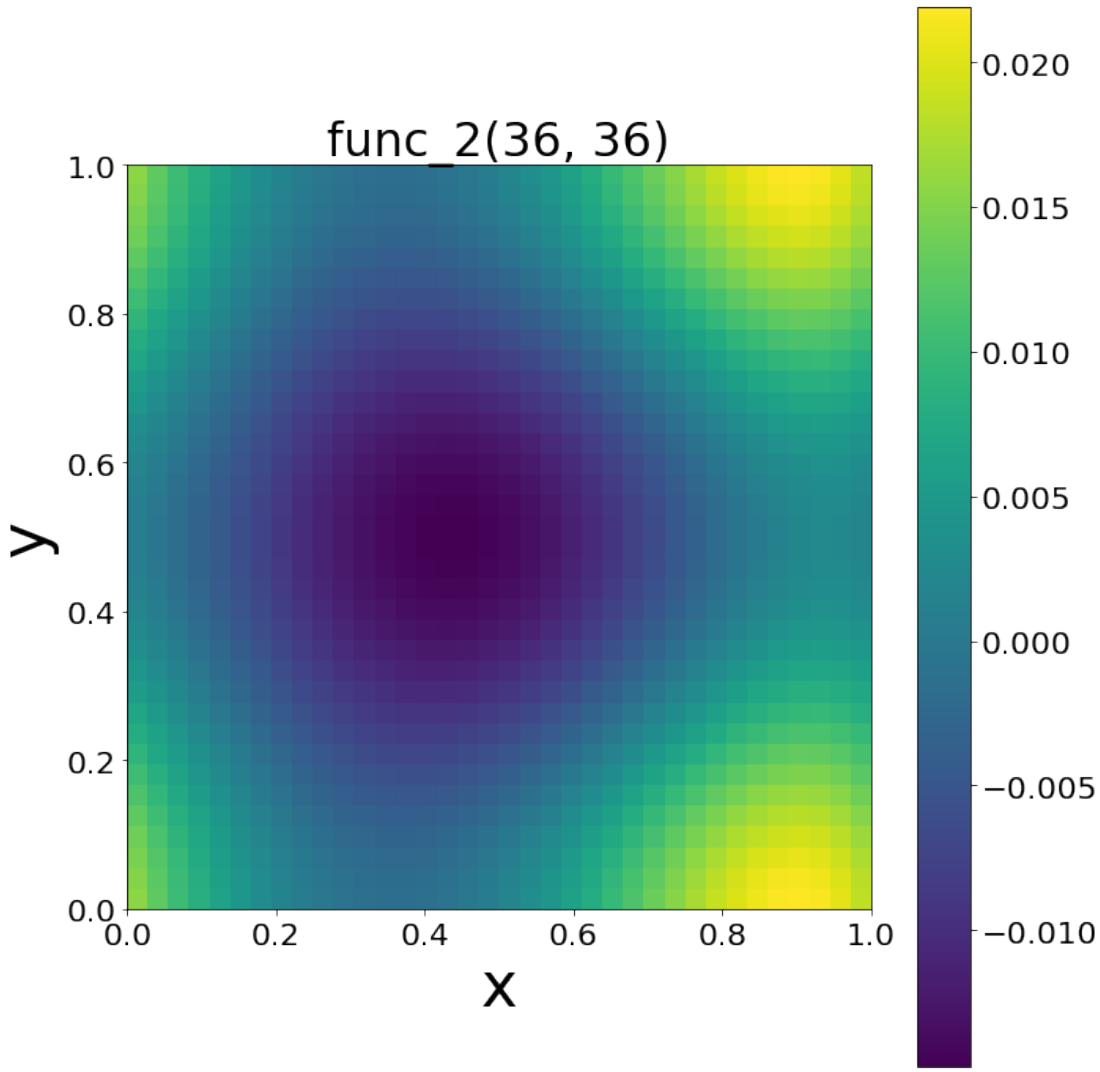


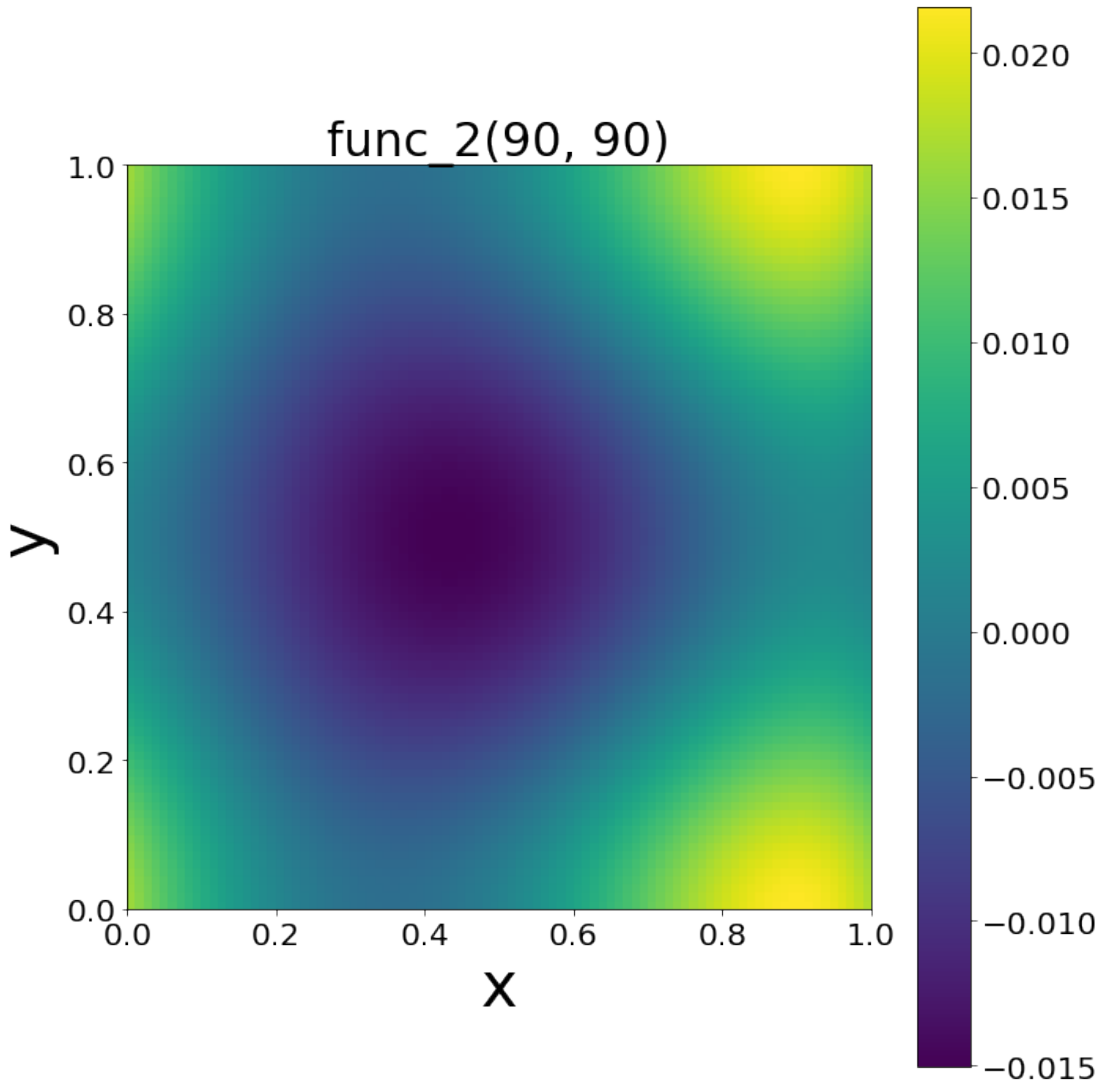
In [13]:
```
Z = np.transpose(solution_1)
trace = go.Surface(z=Z, colorscale='Viridis')
iplot([trace])
```

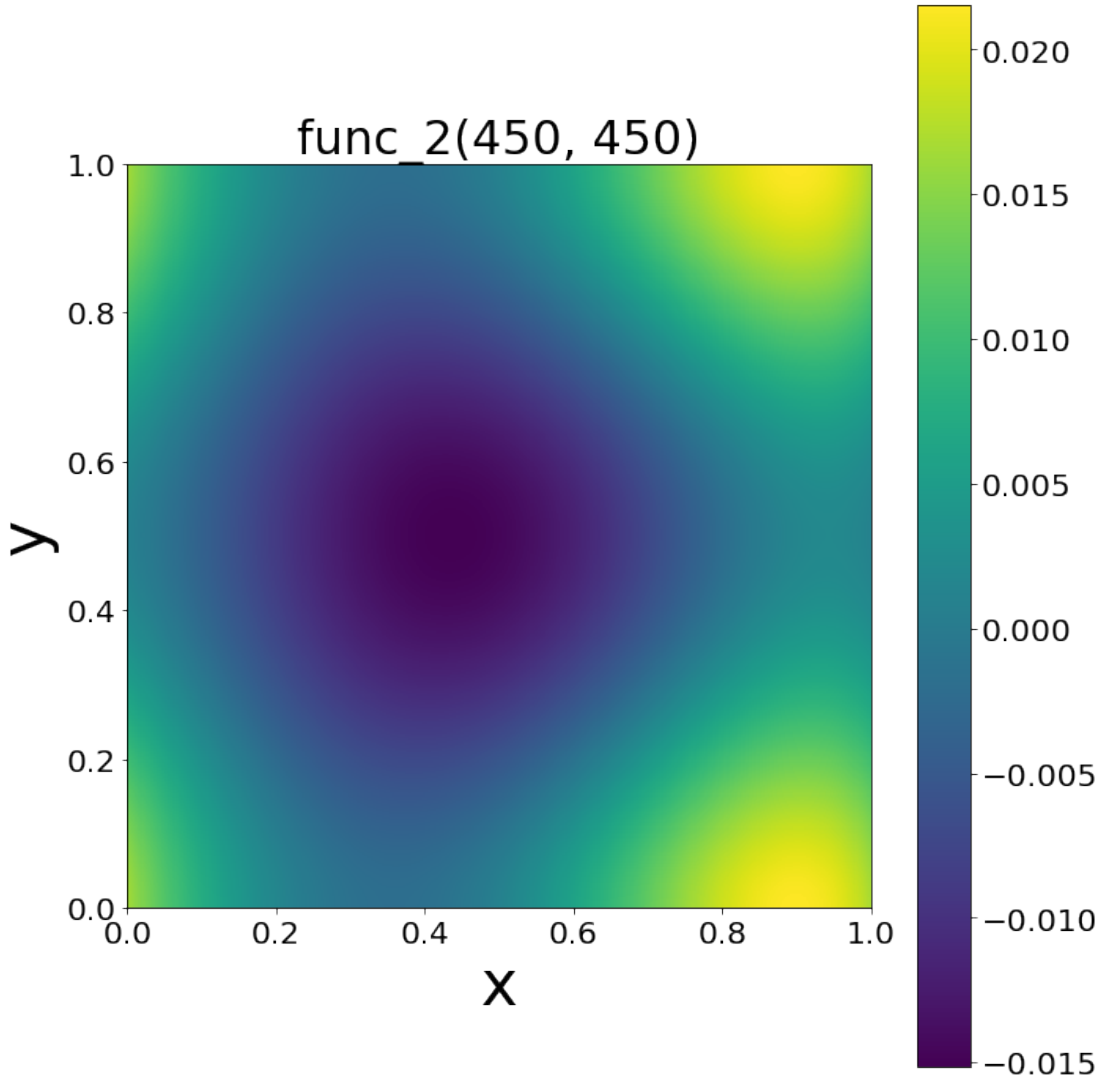
```
In [14]: loss_2 = []
         solution_2, loss_2 = get_results(N, loss_2, func_2, phi_2)

 :
[10, 16, 20, 25, 32, 36, 40, 45, 50, 90, 130, 170, 210, 250, 290, 330, 370, 410, 450, 490]
     :
[0.00409122 0.00439179 0.00446394 0.00451067 0.00454329 0.00455406
 0.00456178 0.00456869 0.00457364 0.00458826 0.00459167 0.00459297
 0.0045936  0.00459395 0.00459417 0.00459432 0.00459442 0.00459449
 0.00459454 0.00459458]
```

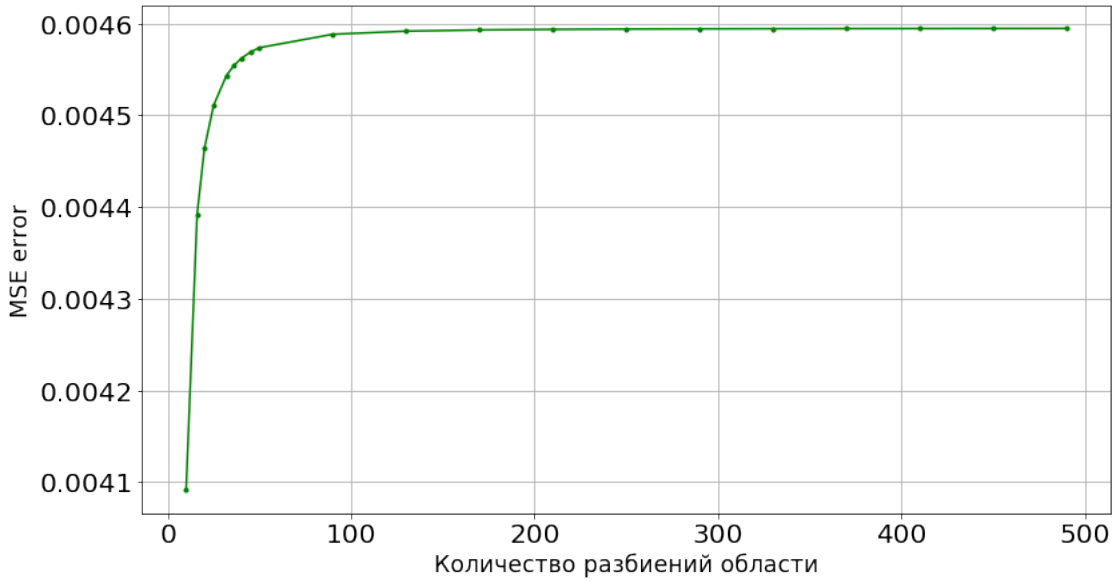



func_2(36, 36)
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
0.020
0.015
0.010
0.005
0.000
−0.005
−0.010

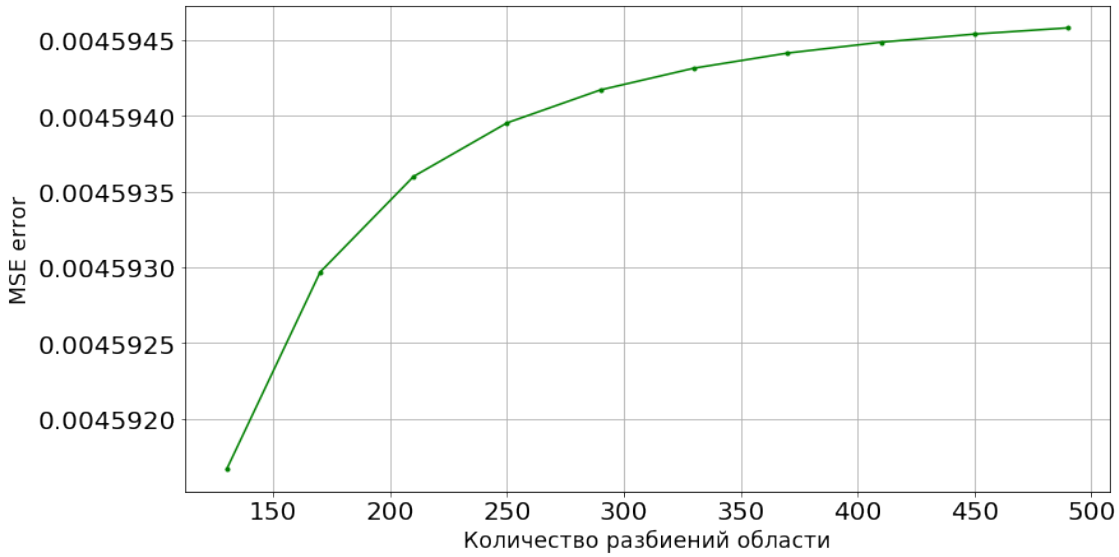

func_2(90, 90)
y
x
0.0
0.2
0.4
0.6
0.8
1.0
0.020
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015



In [16]: `plot_loss(loss_2, N, func_2)`



In [15]:
```
plot_loss(loss_2[10:], N[10:], func_2)
```



In [17]:
```
Z = np.transpose(solution_2)
trace = go.Surface(z=Z, colorscale='Viridis')
iplot([trace])
```