PDE_2

May 21, 2023

1 PDE python exercise 2

In this exercise we will focus on work the manipulation of arrays and 3D plotting. We will use a simple mathematical expression as a background for the development of our python code.

First we import our mandatory modules

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
```

In this exercise we will use a function F which is dependent upon two variables x and y. This equation reads:

$$F = x^2 * y^2 + \sin(x) + \cos(y)$$

In this problem, you need to: 1. x and y must be defined using a `numpy meshgrid` in the range 0 to 1 2. Define a function which computes F with both x and y as input 3. Define a function which computes the partial derivative of F with respect to x 4. Define a function which computes the partial derivative of F with respect to y 5. Plot the resulting function F and its derivatives $\frac{\partial F}{\partial x}$ and $\frac{\partial F}{\partial y}$ using a 3D plot 6. Plot the resulting function F and its derivatives $\frac{\partial F}{\partial x}$ and $\frac{\partial F}{\partial y}$ using a 2D contour plot

In this python exercise, F and its derivatives have to be computed for a given range of x and y. The function must be evaluated as: ´F = func(X,Y)´

Without using a `for` loop:

7. Compute the partial derivates of F with respect to both x and y (1st derivative) using a backward difference.
8. Compute the partial derivates of F with respect to both x and y (1st derivative) using a forward difference.

First we create a numpy meshgrid for x and y:

[ ]:
```
x = np.linspace(0.0,1.0,100)
y = np.linspace(0.0,1.0,100)

[X,Y] = np.meshgrid(x,y)
```

We create a function which computes the value of our function F for a given set of x and y

[ ]:
```
def func(X,Y):
    return X**2.0*Y**2.0+np.sin(X)+np.cos(Y)
```

We create two functions which computes the partial derivatives of our function F with respect to x and y:

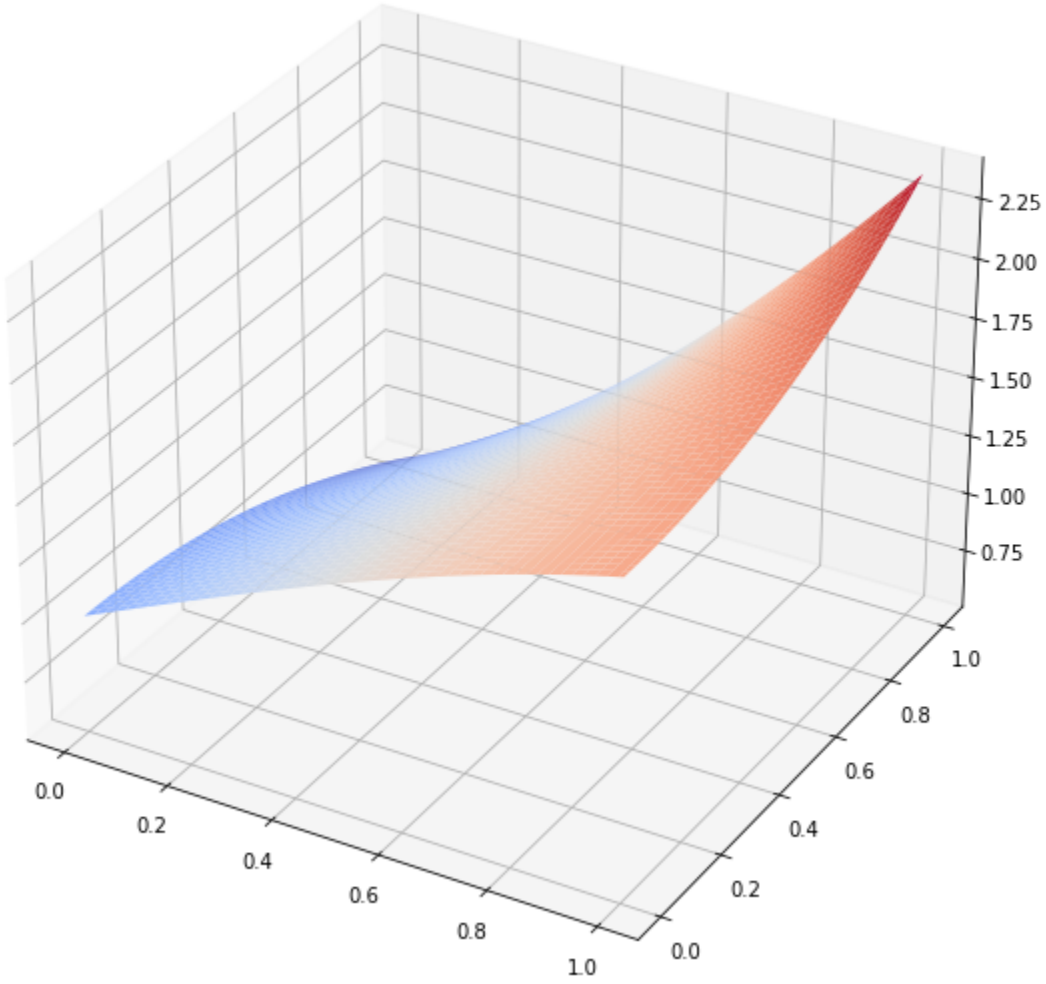
[ ]:
```
def der_func_x(X,Y):
    return 2.0*X*Y**2.0+np.cos(X)

def der_func_y(X,Y):
    return 2.0*Y*X**2.0-np.sin(Y)
```

We then plot our function F using a plot_surface as well as its derivatives in independent plots.

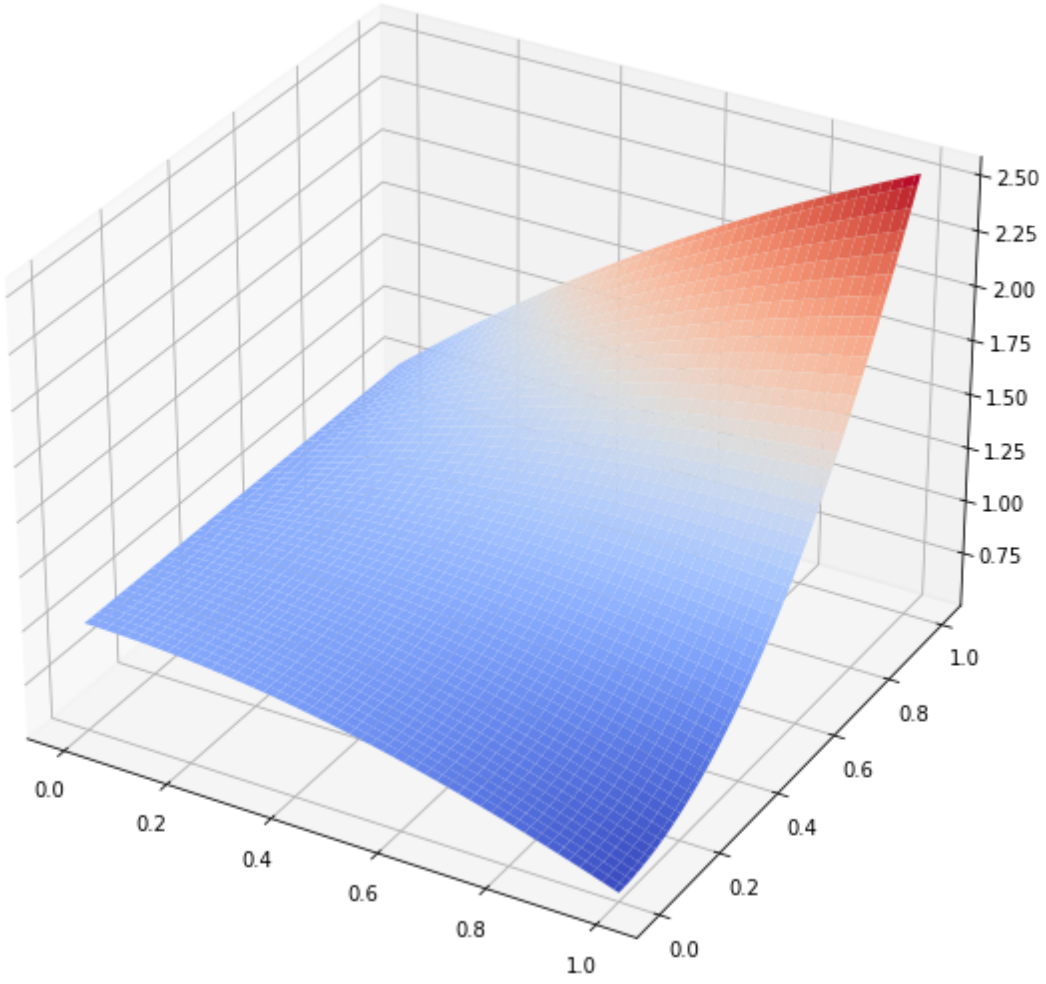
[ ]:
```
F = func(X,Y)
ax = plt.figure(figsize=(10,10)).add_subplot(projection='3d')
ax.plot_surface(X, Y, F,cmap='coolwarm')
```

[ ]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x152e65b80>
```



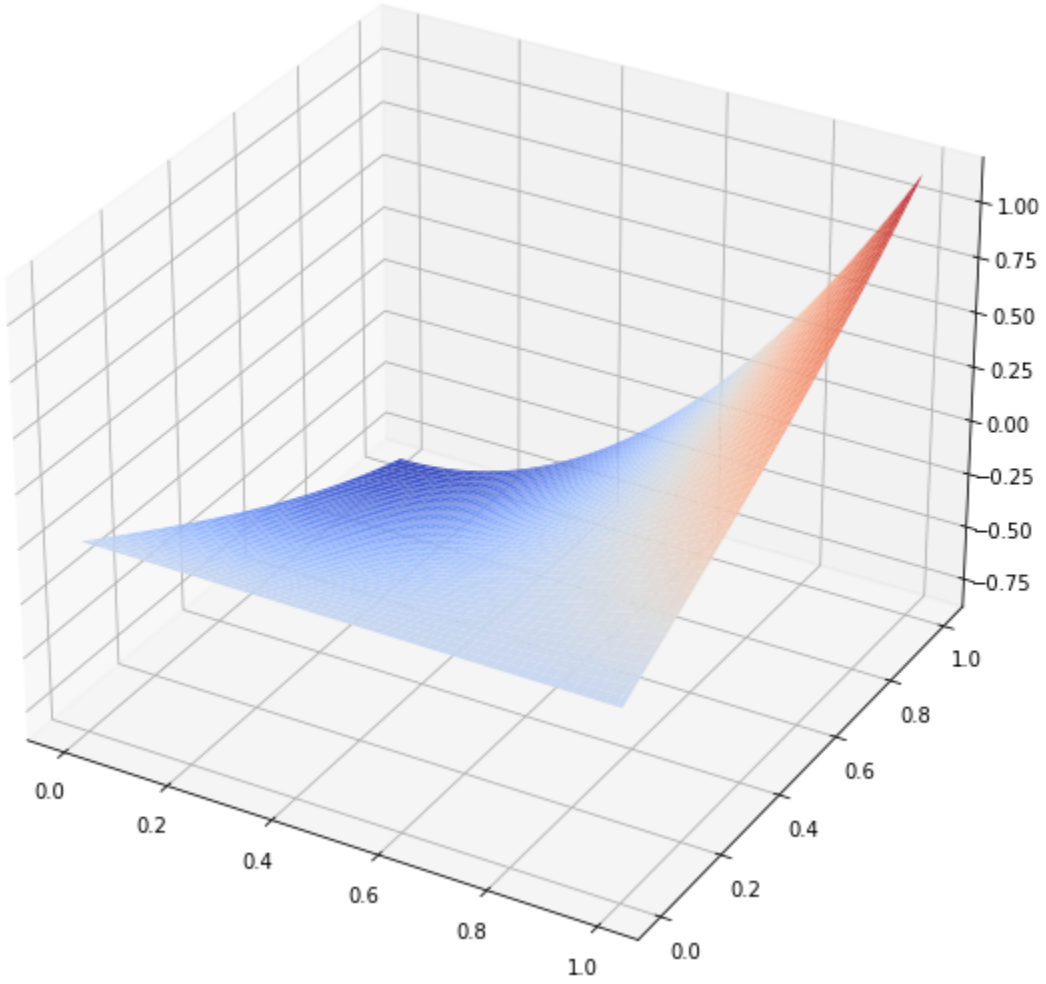
[ ]:
```
ax = plt.figure(figsize=(10,10)).add_subplot(projection='3d')
ax.plot_surface(X, Y, der_func_x(X,Y),cmap='coolwarm')
```

[ ]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1533e6730>
```



[ ]:
```
ax = plt.figure(figsize=(10,10)).add_subplot(projection='3d')
ax.plot_surface(X, Y, der_func_y(X,Y),cmap='coolwarm')
```

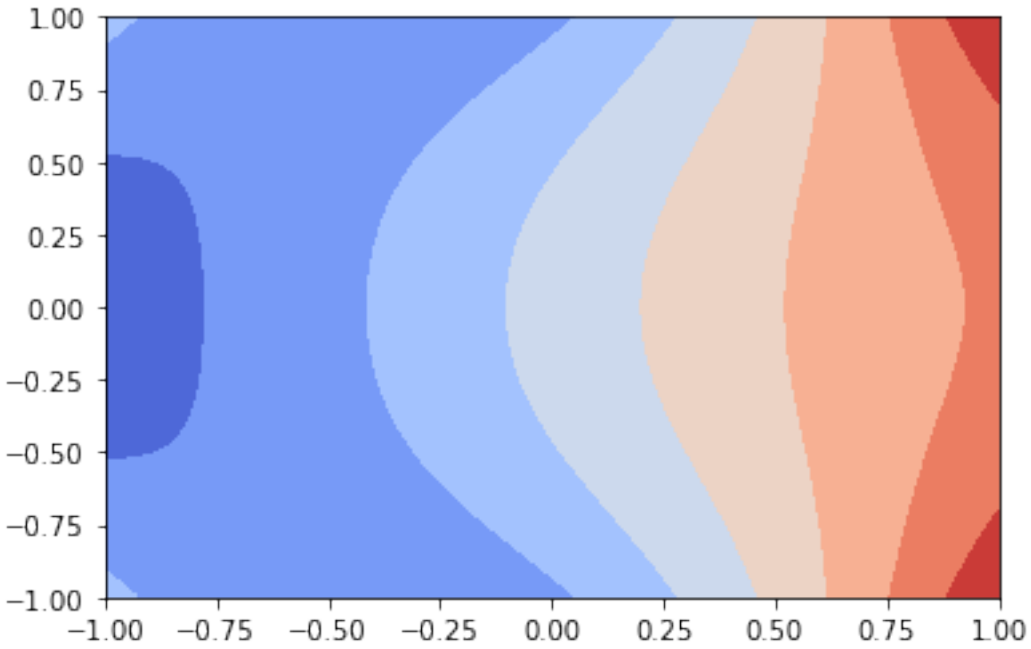
[ ]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x15316ed30>
```



To plot our function F and its derivatives in a contour plot, we use the filled contour module `contourf`

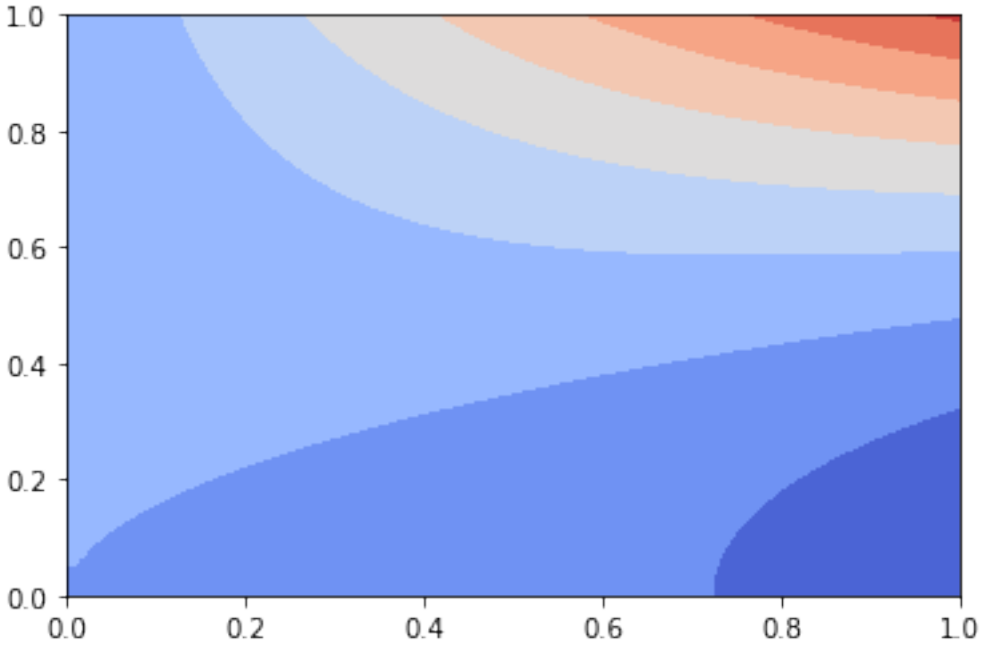
[ ]:
```
plt.contourf(X,Y,F,cmap='coolwarm')
```

[ ]:
```
<matplotlib.contour.QuadContourSet at 0x1690ca6a0>
```



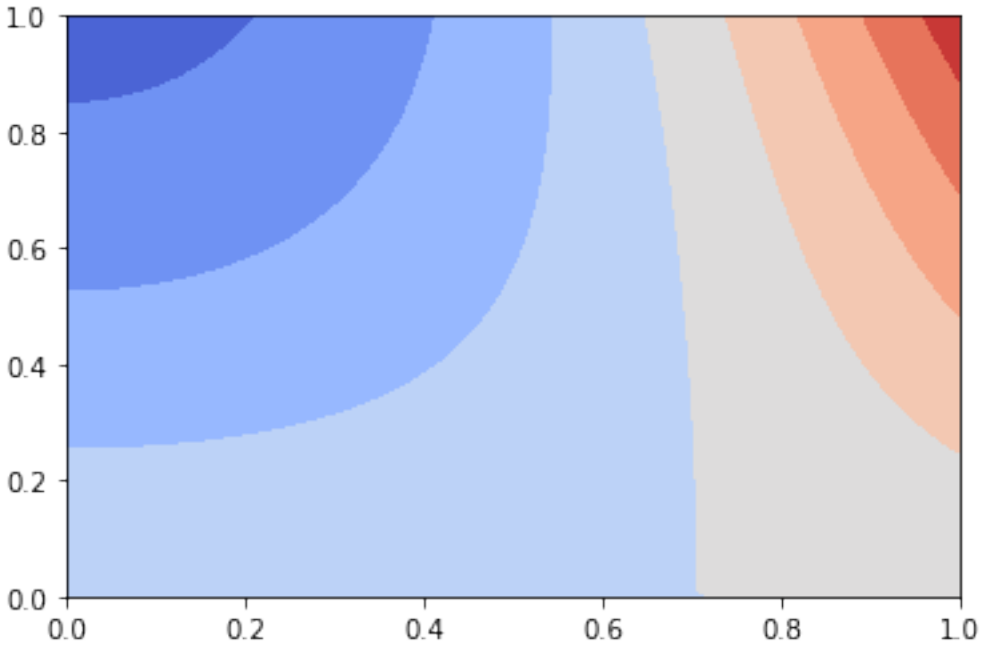
[ ]:
```
plt.contourf(X,Y,der_func_x(X,Y),cmap='coolwarm')
```

[ ]:
```
<matplotlib.contour.QuadContourSet at 0x153af7cd0>
```



[ ]:
```
plt.contourf(X,Y,der_func_y(X,Y),cmap='coolwarm')
```

[ ]:
```
<matplotlib.contour.QuadContourSet at 0x1531eb850>
```



To compute the first derivative of F without using a for loop we need to manipulate arrays.

[ ]:
```
F = func(X,Y)

der_x_BD = (F[:,1:]-F[:,:-1])/(X[:,1:]-X[:,:-1])
der_y_BD = (F[1:,:]-F[:-1,:])/(Y[1:,:]-Y[:-1,:])

der_x_FD = (F[:,1:]-F[:,:-1])/(X[:,1:]-X[:,:-1])
der_y_FD = (F[1:,:]-F[:-1,:])/(Y[1:,:]-Y[:-1,:])

print(np.shape(der_x_BD))
print(np.shape(der_x_FD))
```

```
(100, 99)
(100, 99)
```

Both the forward and backward differences have the same formulation when slicing the arrays. The only difference is linked to which of the starting or ending row is skipped when using the derivatives. If we plot both the backward and forward difference of F with respect to x in a contourplot we must reduce the size of x to match the size of the derivatives. Wherever we shorten x (start or end) will be our only difference.

[ ]:
```
fig,ax = plt.subplots(1,2)
ax[0].contourf(X[:,1:],Y[:,1:],der_x_BD)
ax[1].contourf(X[:,:-1],Y[:,:-1],der_x_FD)
```

[ ]:
```
<matplotlib.contour.QuadContourSet at 0x1562a4580>
```