OPT_3

May 21, 2023

1 Global optimization exercise

In this exercise we want to the minimize the following cost function $f(x, y)$

$$f(x, y) = (x - \sin(2.x.y + x))^2 + (y - \cos(x.y))^2$$

The variables x and y are bounded and defined by:

$$-2 \leq x \leq 4$$
$$-3 \leq y \leq 1$$

In this exercice you should: 1. Define a function to compute the cost function f to be used within the optimization packages. 2. Plot the cost function f as a filled contour plot alongside a colorbar within the bounds define earlier. 3. Using the starting points $x_0 = 6.0$ and $y_0 = 0.0$ and the SLSQP algorithm find the optimal solution of the defined problem. 4. Plot both the initial guess and the optimized value of the parameters onto the countour plot of the cost function. 5. Using the starting points $x_0 = 6.0$ and $y_0 = 0.5$ and the SLSQP algorithm find the optimal solution of the defined problem. 6. Plot both the initial guess and the optimized value of the parameters onto the countour plot of the cost function. Can you comment on what you observe? 7. Repeat the optimization problem using the basin hopping algorithm. You can use the two starting points defined in questions 3 and 4. In the basin hopping algorithm you can use 10 iterations. 8. Repeat question 7 using 20 iterations instead of 10. 9. Apply the differential evolution algorithms to find the global minima of the cost function. 10. Discuss the numerical cost (number of iterations and function evaluations) associated with the different algorithms used in 5, 8 and 9. Can you comment on what you observe?

We start to import our mandatory modules for the exercise.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
```

Question 1

We make a function that define the cost function f of our problem.

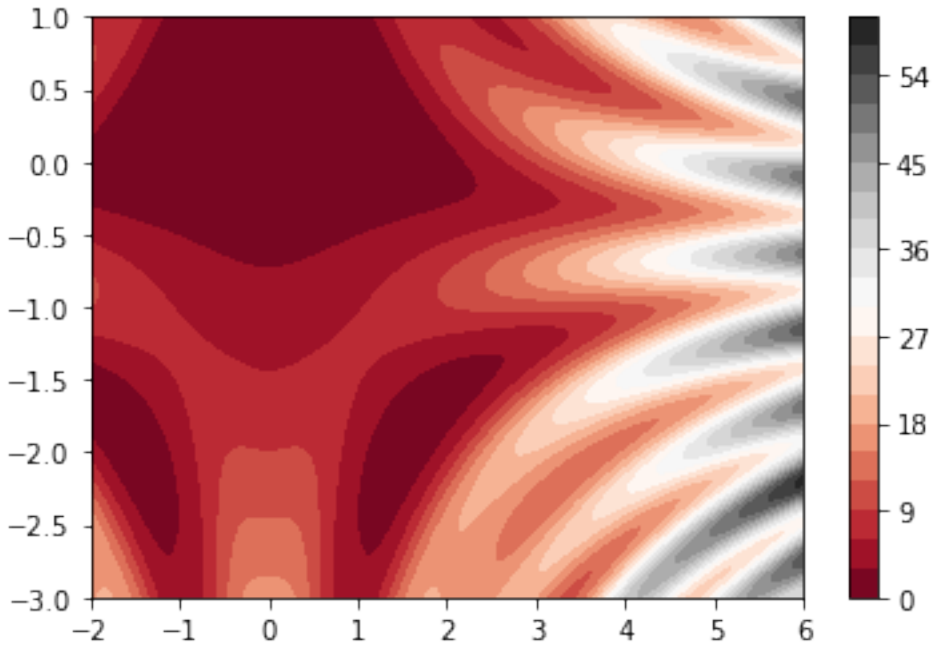
[ ]:
```
def func(data):
    [X,Y] = data
    return (X-np.sin(2*X*Y+X))**2.0+(Y-np.cos(X*Y))**2.0
```

Question 2

we can plot the cost function over the range of allowable values for x and y.

[ ]:
```
X,Y = np.meshgrid(np.linspace(-2.0,6.0,1000),np.linspace(-3.0,1.0,1000))
Z   = func([X,Y])
plt.contourf(X, Y, Z, 20, cmap='RdGy')
plt.colorbar()
```

[ ]:
```
<matplotlib.colorbar.Colorbar at 0x12b596d00>
```



Question 3

We can then define the optimization problem starting with the bounds for the variables x and y. The first optimization is carried out with the starting point being set to $x_0 = 6.0$ and $y_0 = 0.0$.

[ ]:
```
bounds = []
bounds.append([-2.0,4.0])
bounds.append([-3.0,1.0])
```

[ ]:
```
x0 = [6.0,0.0]
options = {'disp': True, 'maxiter':1000}
res = optimize.minimize(func, x0, method='SLSQP',bounds=bounds,options=options)
```

```
x_opt = res.x
n_it_SLSQP,n_fev_SLSQP = [],[]
n_it_SLSQP.append(res.nit)
n_fev_SLSQP.append(res.nfev)
```

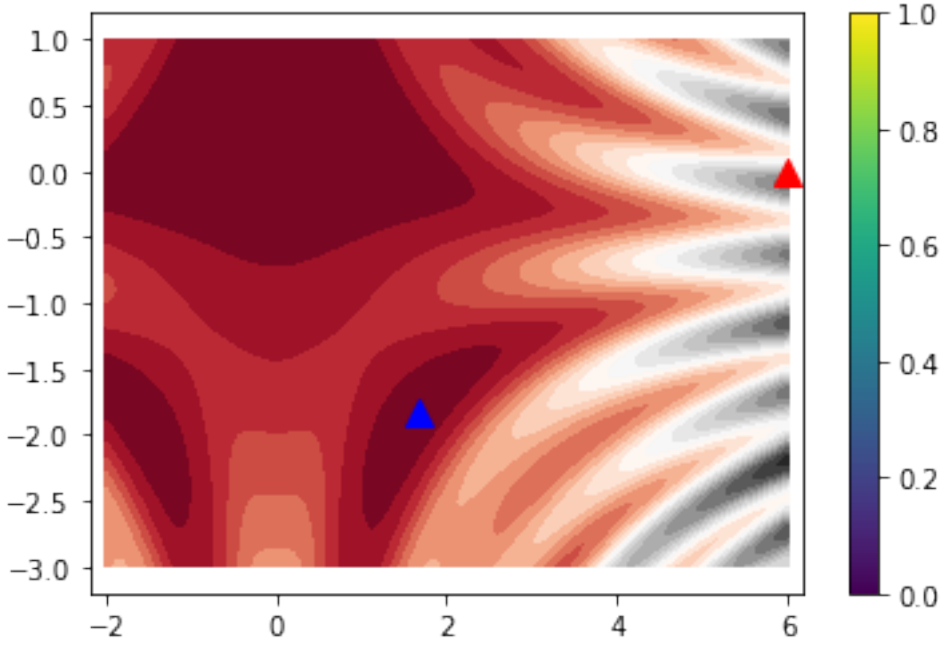
```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 1.193053712920554
            Iterations: 7
            Function evaluations: 24
            Gradient evaluations: 7
```

Question 4

we can plot the starting point and optimum solution onto the contour plot of the cost function.

[ ]:
```
X,Y = np.meshgrid(np.linspace(-2.0,6.0,1000),np.linspace(-3.0,1.0,1000))
Z   = func([X,Y])
plt.contourf(X, Y, Z, 20, cmap='RdGy')
plt.scatter(x0[0],x0[1],c='r',s=100,marker='^')
plt.scatter(x_opt[0],x_opt[1],c='b',s=100,marker='^')
plt.xlim([-2.2,6.2])
plt.ylim([-3.2,1.2])
plt.colorbar()
```

[ ]:
```
<matplotlib.colorbar.Colorbar at 0x14fa10f10>
```



Question 5

we carry out another local minimization using the SLSQP algorithm with the starting point defined as $[x_0 = 6$ and $y = 0.5]$

[ ]:
```
x0 = [6.0,0.5]
options = {'disp': True, 'maxiter':1000}
res = optimize.minimize(func, x0, method='SLSQP',bounds=bounds,options=options)
x_opt = res.x
n_it_SLSQP.append(res.nit)
n_fev_SLSQP.append(res.nfev)
```

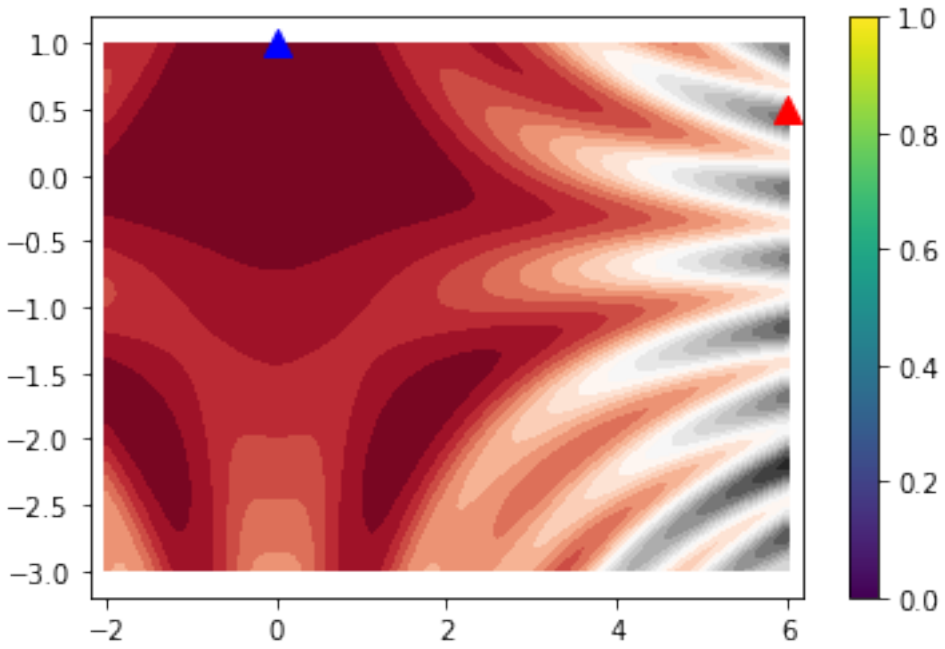
```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 4.670085737083158e-09
            Iterations: 12
            Function evaluations: 39
            Gradient evaluations: 12
```

Question 6

We, again, plot the starting point for the optimization as well as the optimum value and observed a very different result than in the previous question. It thus indicate that our problem is sensitive to local minimas as it can be seen from the cost function plot.

[ ]:
```
X,Y = np.meshgrid(np.linspace(-2.0,6.0,1000),np.linspace(-3.0,1.0,1000))
Z   = func([X,Y])
plt.contourf(X, Y, Z, 20, cmap='RdGy')
plt.scatter(x0[0],x0[1],c='r',s=100,marker='^')
plt.scatter(x_opt[0],x_opt[1],c='b',s=100,marker='^')
plt.xlim([-2.2,6.2])
plt.ylim([-3.2,1.2])
plt.colorbar()
```

[ ]:
```
<matplotlib.colorbar.Colorbar at 0x14fb1c910>
```



Question 7

we now apply a global minimizer to solve our optimization problem. Despite having a global optimizer we see that the optimal values are different and still depend upon the starting points.

[ ]:
```
x0s = [[6.0,0.0],[6.0,0.5]]
x_opt_basinhopping, n_it_basinhopping, n_fev_basinhopping = [], [], []
for x0 in x0s:
    res = optimize.basinhopping(func, x0, niter = 10,␣
 ↪minimizer_kwargs={'method':'SLSQP','bounds':bounds},disp=False)
    x_opt_basinhopping.append(res.x)
    n_it_basinhopping.append(res.nit)
    n_fev_basinhopping.append(res.nfev)

for x in x_opt_basinhopping:
    print('The resulting x is {} and y {}'.format(*x))
```

```
The resulting x is 1.6630562273359333 and y -1.8272135374712217
The resulting x is -7.684248051302578e-07 and y 1.0
```

Question 8

we now increase the number of iterations for the basin hopping algorithm. We do get similar optimum values but the results change if we repeat the same optimization multiple times. This is a result from the algorithm which randomize the starting points of the local minimization. In this context we need a rather large number of iterations to make sure that we reach the global minimum.

[ ]:
```
x0s = [[6.0,0.0],[6.0,0.5]]
x_opt_basinhopping, n_it_basinhopping, n_fev_basinhopping = [], [], []
for x0 in x0s:
    res = optimize.basinhopping(func, x0, niter = 20,␣
 ↪minimizer_kwargs={'method':'SLSQP','bounds':bounds},disp=False)
    x_opt_basinhopping.append(res.x)
    n_it_basinhopping.append(res.nit)
    n_fev_basinhopping.append(res.nfev)

for x in x_opt_basinhopping:
    print('The resulting x is {} and y {}'.format(*x))
```

```
The resulting x is -8.393605694098703e-07 and y 1.0
The resulting x is 2.3465307375132582e-09 and y 1.0
```

Question 9

we now apply the differential evolution algorithm which is supposed to be a global optimization algorithm. Similar to question 8, if we run the algorithm several times we do not necessarily get the same optimum parameters. In this algorithm, default values for the population size (popsize), for instance, might not be enough to reach the global mimimum of our cost function.

[ ]:
```
x_opt_diff_evol, n_it_diff_evol, n_fev_diff_evol = [], [], []
for x0 in x0s:
    res = optimize.differential_evolution(func, bounds)
    x_opt_diff_evol.append(res.x)
    n_it_diff_evol.append(res.nit)
    n_fev_diff_evol.append(res.nfev)

for x in x_opt_diff_evol:
    print('The resulting x is {} and y {}'.format(*x))
```

```
The resulting x is 0.0 and y 1.0
The resulting x is 0.0 and y 1.0
```

Question 10

To compare the computational costs of the different algorithms used in this notebook, we print out the number of iterations and number of function evaluation for each of the algorithms and the different starting points.

We can see that the local minimizers use much fewer function evaluation as well as iterations to reach the local minimums compared to the global optimizers. While the global optimizers should give us the best solution to our problem we have observed that it might not be the case always. Furthermore, the number of function evaluations because very large, in particular for the differential evolution, compared to the local minimizations. Therefore, these global optimizers should be used with caution.

[ ]:
```
print('SLSQP')
for it,fev in zip(n_it_SLSQP,n_fev_SLSQP):
```

```
    print('Number of iterations = {}, number of function evaluation = {}'.
 ↪format(it,fev))

print('Basin hopping')
for it,fev in zip(n_it_basinhopping,n_fev_basinhopping):
    print('Number of iterations = {}, number of function evaluation = {}'.
 ↪format(it,fev))

print('Differential evolution')
for it,fev in zip(n_it_diff_evol,n_fev_diff_evol):
    print('Number of iterations = {}, number of function evaluation = {}'.
 ↪format(it,fev))
```

```
SLSQP
Number of iterations = 7, number of function evaluation = 24
Number of iterations = 12, number of function evaluation = 39
Basin hopping
Number of iterations = 20, number of function evaluation = 497
Number of iterations = 20, number of function evaluation = 426
Differential evolution
Number of iterations = 140, number of function evaluation = 4233
Number of iterations = 125, number of function evaluation = 3783
```