optim_functions

December 1, 2025

#

UP3, Optimization for machine learning: practical session 1: determinist gradient descents

This notebook contains the questions of the practical session along with complementary guidelines and examples. The code is written in Python. The questions are in red.

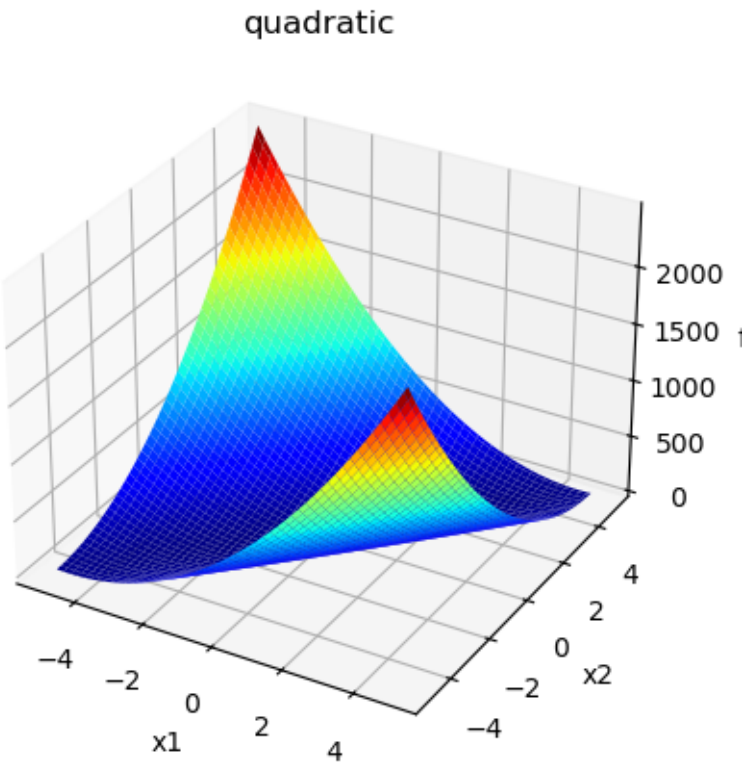
First import all given code.

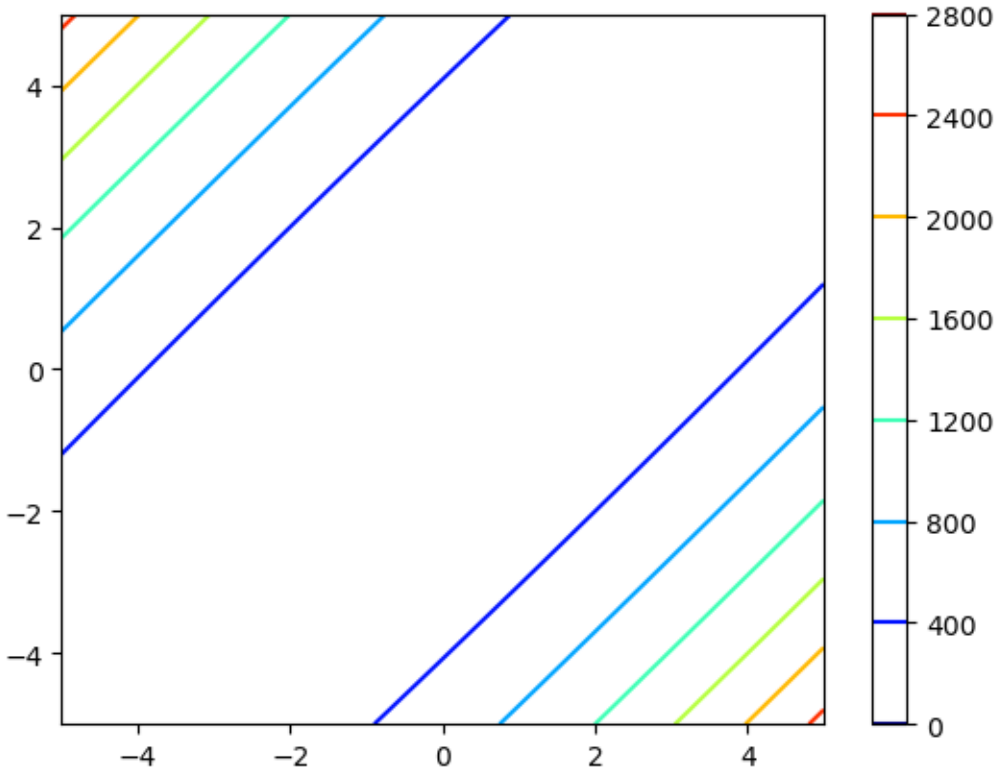
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from typing import Callable, List

from gradient_descent import gradient_descent
from optim_utilities import print_rec
from test_functions import (
    linear_function,
    ackley,
    sphere,
    quadratic,
    rosen,
    L1norm,
    sphereL1,
    rastrigin,
    michalewicz,
    schwefel
)
from restarted_gradient_descent import restarted_gradient_descent
from random_search import random_opt  # always useful to compare optim algos to␣
 ↪a random search

# auto reload to reload functions imported that have been changed (cf.␣
 ↪test_functions.sphereL1 for lbda)
%load_ext autoreload
%autoreload 2
```

0.1 Code demo

Seat and relax, we will show you how to use the code for optimizing functions. First plot examples of 2D functions, which are given in `test_functions.py`

[2]:
```
# function definition
dim = 2
LB = [-5,-5]
UB = [5,5]
fun = quadratic # many other possible functions : cf. test_functions.py


# start drawing the function (necessarily dim==2)
no_grid = 100
#

x1 = np.linspace(start=LB[0], stop=UB[0],num=no_grid)
x2 = np.linspace(start=LB[1], stop=UB[1],num=no_grid)
x, y = np.meshgrid(x1, x2)
xy = np.array([x,y])
z = np.apply_along_axis(fun,0,xy)
figure = plt.figure()
axis = figure.add_subplot(111, projection='3d')
axis.set_zlim(np.min(z)-0.1,np.max(z)+0.1)
axis.plot_surface(x, y, z, cmap='jet', shade= "false")
plt.xlabel(xlabel="x1")
plt.ylabel(ylabel="x2")
plt.title(label=fun.__name__)
axis.set_zlabel("f")
plt.show()
plt.contour(x,y,z,cmap='jet')
plt.colorbar()
plt.show()
# figure.savefig('plot.pdf')
```


quadratic
2000
1500
1000
500
0
f
−4
−2
0
2
4
x1
−4
−2
0
2
4
x2


4
2
0
−2
−4
−4
−2
0
2
4
2800
2400
2000
1600
1200
800
400
0

Now carry out some optimizations.

Some explanations about results format parameters :

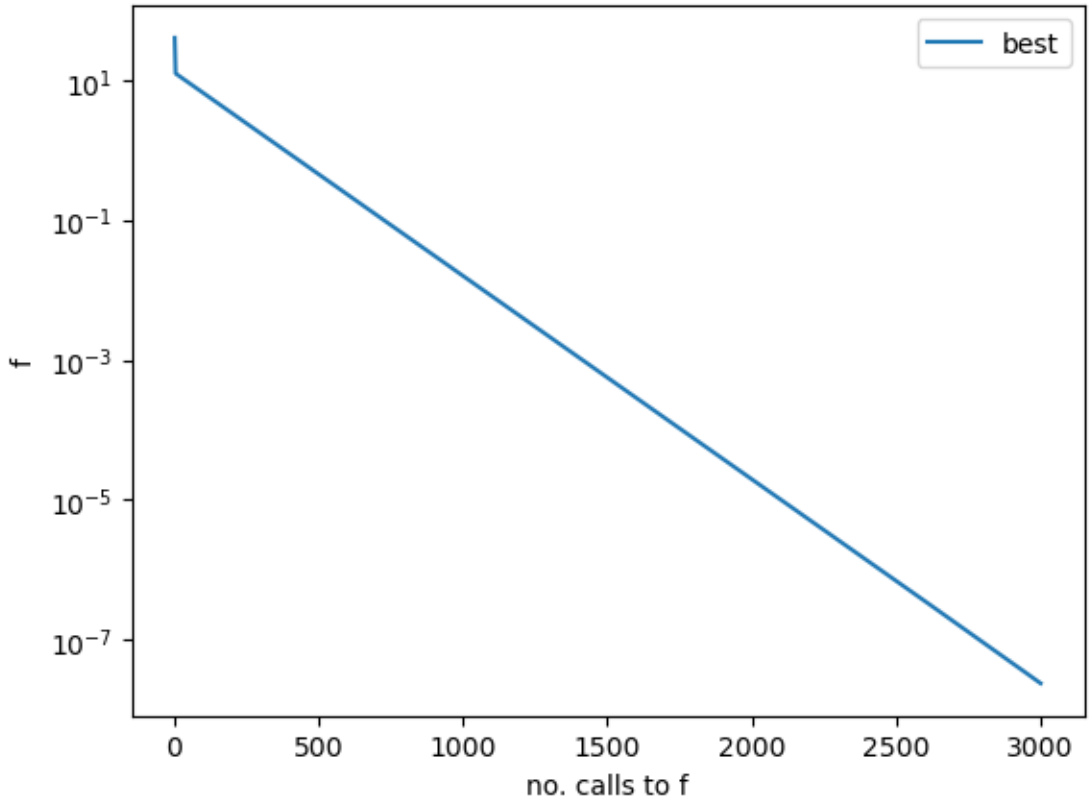
printlevel : int, controls how much is recorded during optimization.
= 0 for minimum recording (best point found and its obj function value)
> 0 records history of best points
> 1 records the entire history of points (memory consuming)

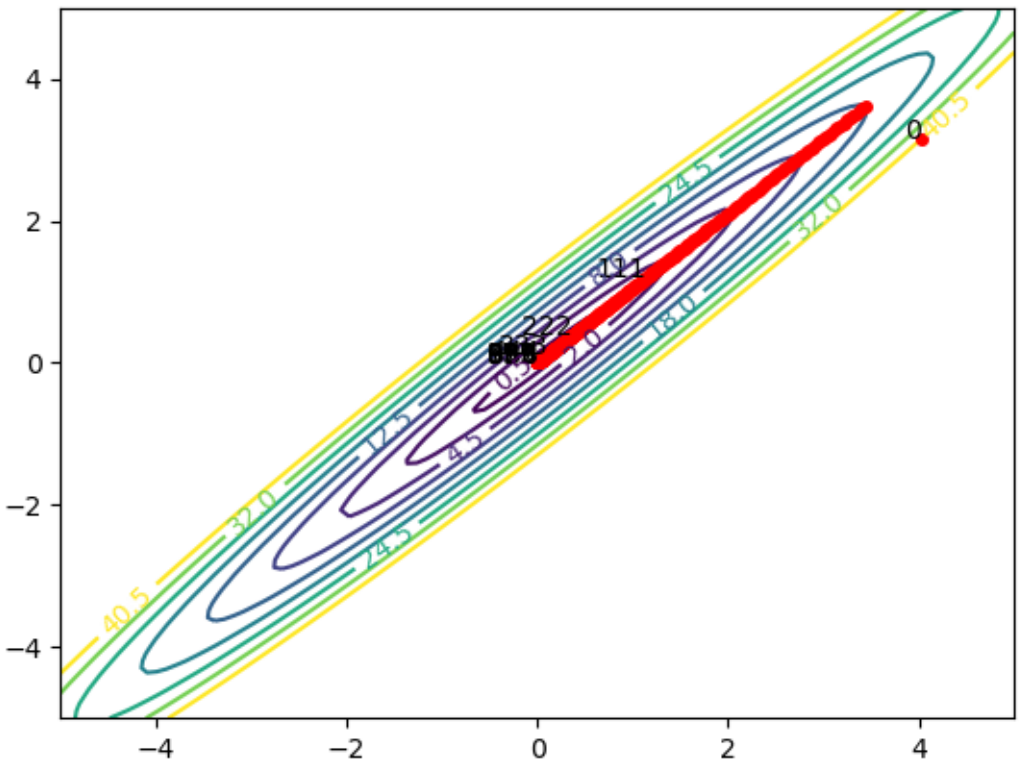
The optimization results are dictionaries with the following key-value pairs:
"f_best", float : best ojective function found during the search
"x_best", 1D array : best point found
"stop_condition" : str describing why the search stopped
"time_used" , int : time actually used by search (may be smaller than max budget)
if printlevel > 0 :
"hist_f_best", list(float) : history of best so far objective functions
"hist_time_best", list(int) : times of recordings of new best so far
"hist_x_best", 2D array : history of best so far points as a matrix, each x is a row
if printlevel > 1 :
"hist_f", list(float) : all f's calculated
"hist_x", 2D array : all x's calculated
"hist_time", list(int) : times of recording of full history

[3]:
```
##########################
# function definition
fun = quadratic
dim = 2
LB = [-5] * dim
UB = [5] * dim
# np.random.seed(123) # useful for repeated runs (quadratic fct or initial␣
 ↪random point)

##########################
# algorithms settings
# start_x = np.array([3,2,1,-4.5,4.6,-2,-1,4.9,0,2])
# start_x = (1+np.arange(dim))*5/dim
# start_x = np.array([2.3,4.5])
start_x = np.random.uniform(low=LB,high=UB)

budget = 1000*(dim+1)
printlevel = 1  # =0,1,2 , careful with 2 which is memory consuming

##########################
# optimize
# res = random_opt(func=fun, LB=LB, UB=UB, budget=budget, printlevel=printlevel)
res = gradient_descent(func=fun,start_x=start_x, LB=LB,UB=UB,budget=budget,
```

```
                        step_factor=0.01,direction_type="gradient",
                        do_linesearch=False,min_step_size=1e-11,
                        min_grad_size=1e-6,inertia=0.9,printlevel=printlevel)

##########################
# reporting
print_rec(res=res, fun=fun, dim=dim, LB=LB, UB=UB , printlevel=printlevel,␣
 ↪logscale = True)
```

```
search stopped after 3000 evaluations of f because of budget exhausted
best objective function = 2.348770125867263e-08
best x = [0.00014936 0.00015705]
```





0.1.1 Question 1: local vs. global search

Change the following features of the above code to observe the difference between local and global optimizations : 1. Set the dimension of the problem to 2 (so as to observe points in the `x`-space), and select alternatively `ackley` and `quadratic` as objective functions 2. Set the initial point of the search `start_x` somewhere between the bounds `LB` and `UB` 3. Compare the pure gradient descent (`step_factor=0.01,direction_type="gradient",do_linesearch=False`) with the random optimization. You can play with the `step_factor` and the `budget`.

0.1.2 Answer 1: local vs. global search

your turn

[4]:
```
# your code here
```

0.1.3 Question 2: gradient vs. momentum vs. NAG

Compare the influence of the directions gradient, momentum and NAG on a quadratic function in relatively high (50) dimensions.

0.1.4 Answer 2: gradient vs. momentum vs. NAG directions

your analysis

[5]:
```
# your code here
```

0.2 A first step towards ML: regularized quadratic loss

Let us consider the following test function which is associated to machine learning :

$$f(x) = \sum_{i=1}^{D} (x_i - c_i)^2 + \lambda \sum_{i=1}^{D} |x_i| \quad , \quad \lambda \geq 0$$

$$c_i = i \quad \text{and} \quad -5 = LB_i \leq x_i \leq UB_i = 5 \quad , \quad i = 1, \ldots, D$$

- First term: sphere function centered at c. A simplistic model to the mean square error of a NN where x minimizes the training error.
- Second term: L1 norm times λ. The x_i's would be the weights of a NN. This term helps in improving the test error.

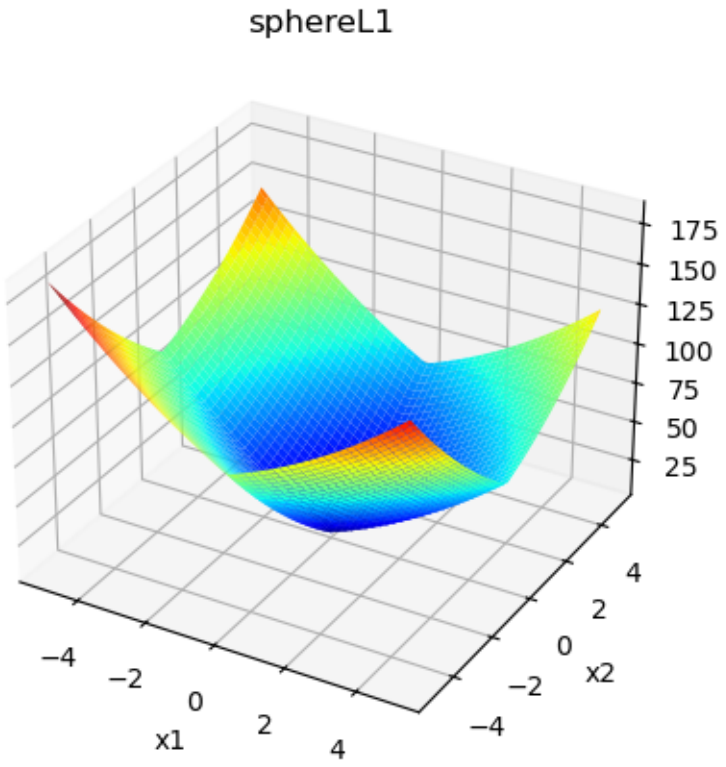
The function is already coded in `test_functions.py` as `sphereL1`. λ is set in the function (open the file in your preferred Python editor).

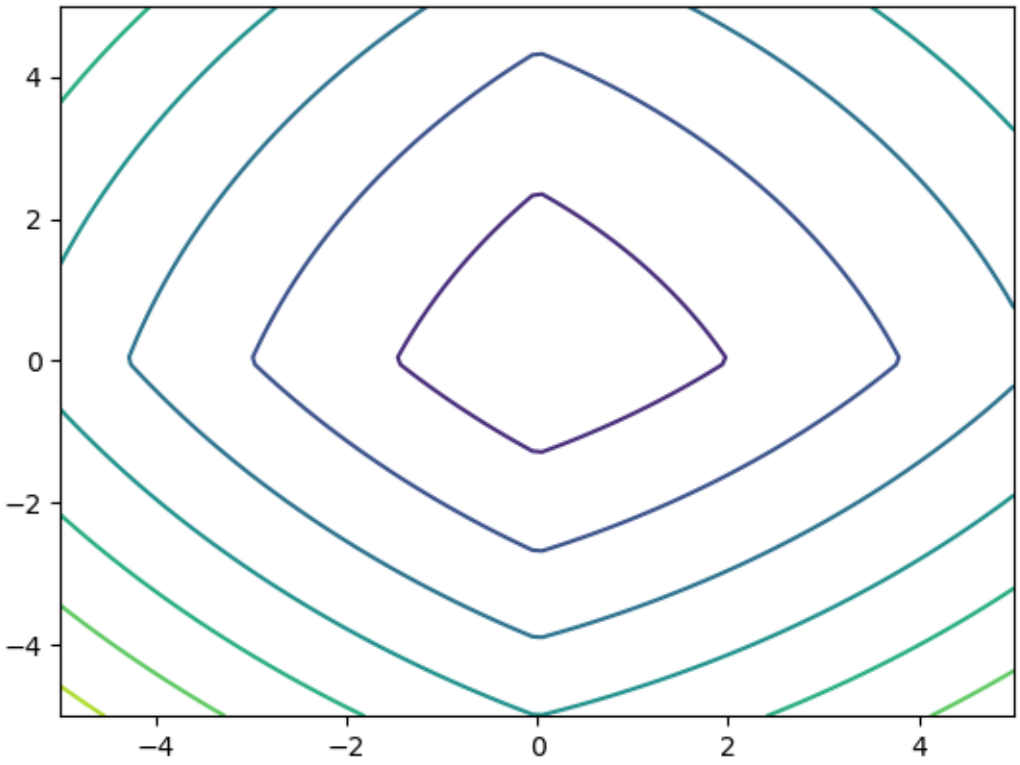
Let us first plot the function in 2 dimensions:

[6]:
```
%load_ext autoreload
%autoreload 2
# function definition
dim = 2
LB = [-5,-5]
UB = [5,5]
fun = sphereL1


# start drawing the function (necessarily dim==2)
no_grid = 100
#
# execute " %matplotlib qt5 " in the spyder console for interactive 3D plots
# " %matplotlib inline " will get back to normal docking
x1 = np.linspace(start=LB[0], stop=UB[0],num=no_grid)
x2 = np.linspace(start=LB[1], stop=UB[1],num=no_grid)
x, y = np.meshgrid(x1, x2)
xy = np.array([x,y])
z = np.apply_along_axis(fun,0,xy)
figure = plt.figure()
axis = figure.add_subplot(111, projection='3d')
axis.plot_surface(x, y, z, cmap='jet', shade= "false")
plt.xlabel(xlabel="x1")
plt.ylabel(ylabel="x2")
plt.title(label=fun.__name__)
axis.set_zlabel("f")
plt.show()
plt.contour(x,y,z)
```

```
plt.show()
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```





0.2.1 Question 3 : optimizing the `sphereL1` function

You will optimize the `sphereL1` function for various values of λ, $\lambda = \{0.001, 0.1, 1, 5, 10\}$ in `dim=10` dimensions.

To do this, make sure that the function is described as follows

```
# function definition
fun = sphereL1
dim = 10
LB = [-5] * dim
UB = [5] * dim
```

Also, to make the results of the search more robust, use the `restarted_gradient_descent` function (cf. `restarted_gradient_descent.py` file).

Repeat optimizations for varying λ's (parameter `lbda` dans `test_functions.sphereL1`) 3.1. What do you notice ? 3.2. Assuming the x's are weights of a neural network, what would be the effect of λ on the network ?

Note : when changing `lbda`, it is important to restart the kernel or, to make it automatic, the following lines of code have been added at the top of the notebook.

```
%load_ext autoreload
%autoreload 2
```

0.2.2 Answer 3: optimizing the `sphereL1` function

your analysis

[7]:
```
# your code here
```

0.3 End of this lab