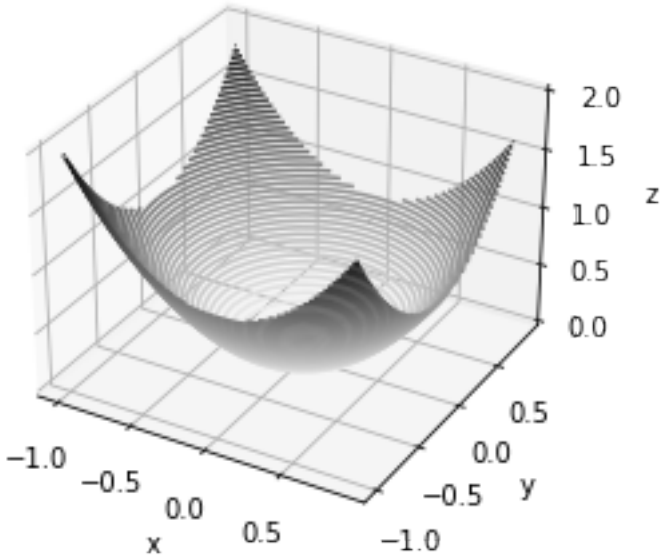
prac2

April 2, 2023

[ ]:
```
def objective(x, y):
    return x**2.0 + y**2.0
```

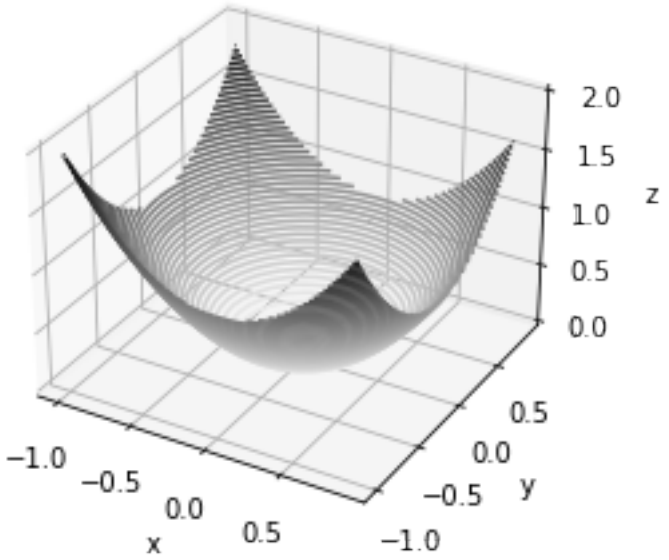
[ ]:
```
from numpy import arange
from numpy import meshgrid
from matplotlib import pyplot

def objective(x, y):
     return x**2.0 + y**2.0

r_min, r_max = -1.0, 1.0
xaxis = arange(r_min, r_max, 0.1)
yaxis = arange(r_min, r_max, 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
figure = pyplot.figure()
ax= pyplot.axes(projection = '3d')
ax.contour3D(x , y , results , 50 , cmap ='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
pyplot.show()
```



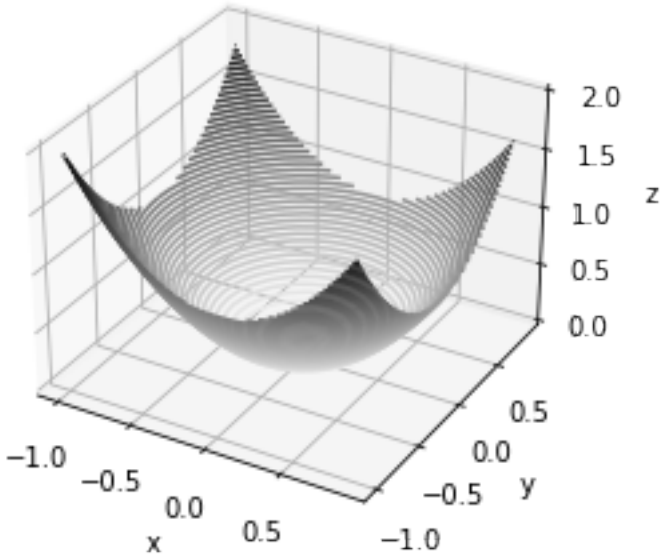
[ ]:
```
from math import sqrt
from numpy import asarray
from numpy import arange
from numpy.random import rand
from numpy.random import seed

def objective(x, y):
    return x**2.0 + y**2.0

def derivative(x, y):
    return asarray([x * 2.0, y * 2.0])

from math import sqrt
from numpy import asarray
from numpy.random import rand
from numpy.random import seed

def objective(x, y):
        return x**2.0 + y**2.0

def derivative(x, y):
        return asarray([x * 2.0, y * 2.0])

def adagrad(objective, derivative, bounds, n_iter, step_size):
        solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:,␣
 ↪0])
```

```
        sq_grad_sums = [0.0 for _ in range(bounds.shape[0])]
        for it in range(n_iter):
                gradient = derivative(solution[0], solution[1])
                for i in range(gradient.shape[0]):
                        sq_grad_sums[i] += gradient[i]**2.0
                new_solution = list()
                for i in range(solution.shape[0]):
                        alpha = step_size / (1e-8 + sqrt(sq_grad_sums[i]))
                        value = solution[i] - alpha * gradient[i]
                        new_solution.append(value)
                solution = asarray(new_solution)
                solution_eval = objective(solution[0], solution[1])
                print('>%d f(%s) = %.5f' % (it, solution, solution_eval))
                r_min, r_max = -1.0, 1.0
                xaxis = arange(r_min, r_max, 0.1)
                yaxis = arange(r_min, r_max, 0.1)
                x, y = meshgrid(xaxis, yaxis)
                results = objective(x, y)
                figure = pyplot.figure()
                ax= pyplot.axes(projection = '3d')
                ax.contour3D(x , y , results , 50 , cmap ='binary')
                ax.set_xlabel('x')
                ax.set_ylabel('y')
                ax.set_zlabel('z')
                pyplot.show()
        return [solution, solution_eval]

seed(1)
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
n_iter = 50
step_size = 0.1
best, score = adagrad(objective, derivative, bounds, n_iter, step_size)
xaxis = arange(bounds[0,0], bounds[0,1], 0.1)
yaxis = arange(bounds[1,0], bounds[1,1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
solutions = asarray(solutions)
pyplot.plot(solutions[:, 0], solutions[:, 1], '.-', color='w')
pyplot.show()
print('Done!')
print('f(%s) = %f' % (best, score))
```

```
>0 f([-0.06595599  0.34064899]) = 0.12039
```

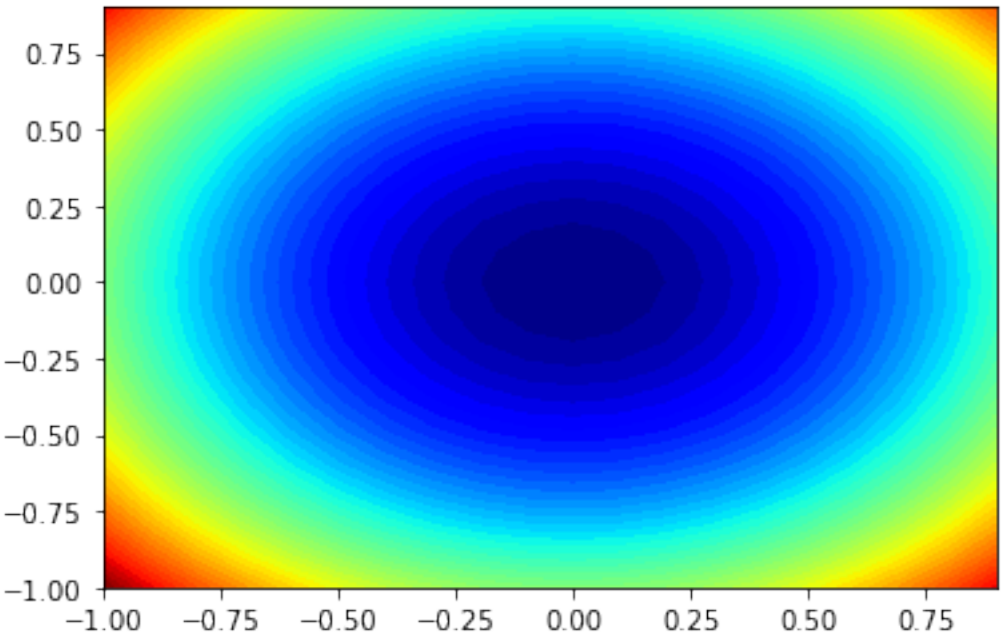



```
>1 f([-0.02902286  0.27948766]) = 0.07896
```




```
>2 f([-0.0129815   0.23463749]) = 0.05522
```

```
     62 pyplot.contourf(x, y, results, levels=50, cmap='jet')
---> 63 solutions = asarray(solutions)
     64 pyplot.plot(solutions[:, 0], solutions[:, 1], '.-', color='w')
     65 pyplot.show()

NameError: name 'solutions' is not defined
```



[ ]:
```python
from math import sqrt
from numpy import asarray
from numpy import arange
from numpy.random import rand
from numpy.random import seed
from numpy import meshgrid
from matplotlib import pyplot
from mpl_toolkits.mplot3d import Axes3D

def objective(x, y):
        return x**2.0 + y**2.0

def derivative(x, y):
        return asarray([x * 2.0, y * 2.0])

def adagrad(objective, derivative, bounds, n_iter, step_size, decay_rate):
        solutions = list()
```

```
        solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:,␣
 ↪0])
        sq_grad_sums = [0.0 for _ in range(bounds.shape[0])]
        for it in range(n_iter):
                gradient = derivative(solution[0], solution[1])
                for i in range(gradient.shape[0]):
                        sq_grad_sums[i] += gradient[i]**2.0
                new_solution = list()
                for i in range(solution.shape[0]):
                        alpha = step_size*decay_rate / (1e-8 +␣
 ↪sqrt(sq_grad_sums[i]))*(1-decay_rate)
                        value = solution[i] - alpha * gradient[i]
                        new_solution.append(value)
                solution = asarray(new_solution)
                solutions.append(solution)
                solution_eval = objective(solution[0], solution[1])
                print('>%d f(%s) = %.5f' % (it, solution, solution_eval))
        return solutions

seed(1)
bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])
decay_rate = 0.3
n_iter = 1000
step_size = 0.1
solutions = adagrad(objective, derivative, bounds, n_iter, step_size ,␣
 ↪decay_rate)
xaxis = arange(bounds[0,0], bounds[0,1], 0.1)
yaxis = arange(bounds[1,0], bounds[1,1], 0.1)
x, y = meshgrid(xaxis, yaxis)
results = objective(x, y)
pyplot.contourf(x, y, results, levels=50, cmap='jet')
solutions = asarray(solutions)
pyplot.plot(solutions[:, 0], solutions[:, 1], '.-', color='w')
pyplot.show()
```

```
>0 f([-0.14495599  0.41964899]) = 0.19712
>1 f([-0.13114119  0.40516653]) = 0.18136
>2 f([-0.12040117  0.39352782]) = 0.16936
>3 f([-0.11147567  0.38357399]) = 0.15956
>4 f([-0.10378581  0.37476647]) = 0.15122
>5 f([-0.0970094   0.36680379]) = 0.14396
>6 f([-0.09094527  0.35949723]) = 0.13751
>7 f([-0.08545775  0.35271944]) = 0.13171
>8 f([-0.08045008  0.34637971]) = 0.12645
>9 f([-0.07585033  0.34041072]) = 0.12163
>10 f([-0.07160319  0.33476088]) = 0.11719
>11 f([-0.067665    0.32938963]) = 0.11308
```

```
>12 f([-0.06400051  0.32426438]) = 0.10924
>13 f([-0.06058074  0.31935849]) = 0.10566
>14 f([-0.05738148  0.31464984]) = 0.10230
>15 f([-0.05438224  0.31011984]) = 0.09913
>16 f([-0.05156546  0.30575267]) = 0.09614
>17 f([-0.04891591  0.30153475]) = 0.09332
>18 f([-0.04642032  0.29745429]) = 0.09063
>19 f([-0.04406696  0.29350102]) = 0.08808
>20 f([-0.04184545  0.28966589]) = 0.08566
>21 f([-0.03974649  0.2859409 ]) = 0.08334
>22 f([-0.03776174  0.28231891]) = 0.08113
>23 f([-0.03588366  0.27879355]) = 0.07901
>24 f([-0.03410539  0.27535909]) = 0.07699
>25 f([-0.03242069  0.27201034]) = 0.07504
>26 f([-0.03082385  0.26874262]) = 0.07317
>27 f([-0.02930961  0.26555163]) = 0.07138
>28 f([-0.02787313  0.26243349]) = 0.06965
>29 f([-0.02650994  0.25938462]) = 0.06798
>30 f([-0.02521588  0.25640172]) = 0.06638
>31 f([-0.0239871   0.25348176]) = 0.06483
>32 f([-0.02282     0.25062195]) = 0.06333
>33 f([-0.02171123  0.24781969]) = 0.06189
>34 f([-0.02065767  0.24507258]) = 0.06049
>35 f([-0.01965637  0.24237837]) = 0.05913
>36 f([-0.01870459  0.23973497]) = 0.05782
>37 f([-0.01779973  0.23714044]) = 0.05655
>38 f([-0.01693937  0.23459293]) = 0.05532
>39 f([-0.01612122  0.23209075]) = 0.05413
>40 f([-0.01534312  0.22963228]) = 0.05297
>41 f([-0.01460303  0.227216  ]) = 0.05184
>42 f([-0.01389904  0.2248405 ]) = 0.05075
>43 f([-0.01322933  0.22250442]) = 0.04968
>44 f([-0.01259218  0.22020649]) = 0.04865
>45 f([-0.01198597  0.21794552]) = 0.04764
>46 f([-0.01140916  0.21572035]) = 0.04667
>47 f([-0.01086029  0.21352993]) = 0.04571
>48 f([-0.010338   0.2113732]) = 0.04479
>49 f([-0.00984096  0.20924921]) = 0.04388
>50 f([-0.00936793  0.20715702]) = 0.04300
>51 f([-0.00891775  0.20509576]) = 0.04214
>52 f([-0.00848929  0.20306457]) = 0.04131
>53 f([-0.0080815   0.20106266]) = 0.04049
>54 f([-0.00769336  0.19908925]) = 0.03970
>55 f([-0.00732391  0.19714362]) = 0.03892
>56 f([-0.00697226  0.19522506]) = 0.03816
>57 f([-0.00663754  0.1933329 ]) = 0.03742
>58 f([-0.00631892  0.19146649]) = 0.03670
>59 f([-0.00601562  0.18962522]) = 0.03599
```

```
>972 f([-1.97586462e-22  1.10680661e-04]) = 0.00000
>973 f([-1.88112348e-22  1.09791497e-04]) = 0.00000
>974 f([-1.79092511e-22  1.08909476e-04]) = 0.00000
>975 f([-1.70505168e-22  1.08034542e-04]) = 0.00000
>976 f([-1.62329581e-22  1.07166636e-04]) = 0.00000
>977 f([-1.54546006e-22  1.06305703e-04]) = 0.00000
>978 f([-1.47135648e-22  1.05451686e-04]) = 0.00000
>979 f([-1.40080611e-22  1.04604529e-04]) = 0.00000
>980 f([-1.33363857e-22  1.03764179e-04]) = 0.00000
>981 f([-1.26969166e-22  1.02930579e-04]) = 0.00000
>982 f([-1.20881096e-22  1.02103677e-04]) = 0.00000
>983 f([-1.15084944e-22  1.01283417e-04]) = 0.00000
>984 f([-1.09566712e-22  1.00469747e-04]) = 0.00000
>985 f([-1.04313076e-22  9.96626138e-05]) = 0.00000
>986 f([-9.93113466e-23  9.88619647e-05]) = 0.00000
>987 f([-9.45494465e-23  9.80677477e-05]) = 0.00000
>988 f([-9.00158758e-23  9.72799110e-05]) = 0.00000
>989 f([-8.56996862e-23  9.64984036e-05]) = 0.00000
>990 f([-8.15904544e-23  9.57231745e-05]) = 0.00000
>991 f([-7.76782571e-23  9.49541732e-05]) = 0.00000
>992 f([-7.39536465e-23  9.41913498e-05]) = 0.00000
>993 f([-7.04076280e-23  9.34346546e-05]) = 0.00000
>994 f([-6.70316382e-23  9.26840384e-05]) = 0.00000
>995 f([-6.38175245e-23  9.19394523e-05]) = 0.00000
>996 f([-6.07575250e-23  9.12008479e-05]) = 0.00000
>997 f([-5.78442500e-23  9.04681772e-05]) = 0.00000
>998 f([-5.50706643e-23  8.97413925e-05]) = 0.00000
>999 f([-5.24300698e-23  8.90204464e-05]) = 0.00000
```