**Experiment – 7**

**Adam Optimization Algorithm**

**Source Code:**

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

# objective function

def objective(x, y):

    return x\*\*2.0 + y\*\*2.0

# derivative of objective function

def derivative(x, y):

    return asarray([x \* 2.0, y \* 2.0])

# gradient descent algorithm with adam

def adam(objective, derivative, bounds, n\_iter, alpha, beta1, beta2, eps=1e-8):

    solutions = []

    # generate an initial point

    x = bounds[:, 0] + rand(len(bounds)) \* (bounds[:, 1] - bounds[:, 0])

    score = objective(x[0], x[1])

    # initialize first and second moments

    m = [0.0 for \_ in range(bounds.shape[0])]

    v = [0.0 for \_ in range(bounds.shape[0])]

    # run the gradient descent updates

    for t in range(n\_iter):

        # calculate gradient g(t)

        g = derivative(x[0], x[1])

        # build a solution one variable at a time

        for i in range(bounds.shape[0]):

            # m(t) = beta1 \* m(t-1) + (1 - beta1) \* g(t)

            m[i] = beta1 \* m[i] + (1.0 - beta1) \* g[i]

            # v(t) = beta2 \* v(t-1) + (1 - beta2) \* g(t)^2

            v[i] = beta2 \* v[i] + (1.0 - beta2) \* g[i]\*\*2

            # mhat(t) = m(t) / (1 - beta1(t))

            mhat = m[i] / (1.0 - beta1\*\*(t+1))

            # vhat(t) = v(t) / (1 - beta2(t))

            vhat = v[i] / (1.0 - beta2\*\*(t+1))

            # x(t) = x(t-1) - alpha \* mhat(t) / (sqrt(vhat(t)) + ep)

            x[i] = x[i] - alpha \* mhat / (sqrt(vhat) + eps)

        # evaluate candidate point

        score = objective(x[0], x[1])

        # keep track of solutions

        solutions.append(x.copy())

        # report progress

        print('>%d f(%s) = %.5f' % (t, x, score))

    return solutions

# seed the pseudo random number generator

seed(1)

# define range for input

bounds = asarray([[-1.0, 1.0], [-1.0, 1.0]])

# define the total iterations

n\_iter = 60

# steps size

alpha = 0.02

# factor for average gradient

beta1 = 0.8

# factor for average squared gradient

beta2 = 0.999

# perform the gradient descent search with adam

solutions = adam(objective, derivative, bounds, n\_iter, alpha, beta1, beta2)

# sample input range uniformly at 0.1 increments

xaxis = arange(bounds[0,0], bounds[0,1], 0.1)

yaxis = arange(bounds[1,0], bounds[1,1], 0.1)

# create a mesh from the axis

x, y = meshgrid(xaxis, yaxis)

# compute targets

results = objective(x, y)

# create a filled contour plot with 50 levels and jet color scheme

pyplot.contourf(x, y, results, levels=50, cmap='jet')

# plot the sample as black circles

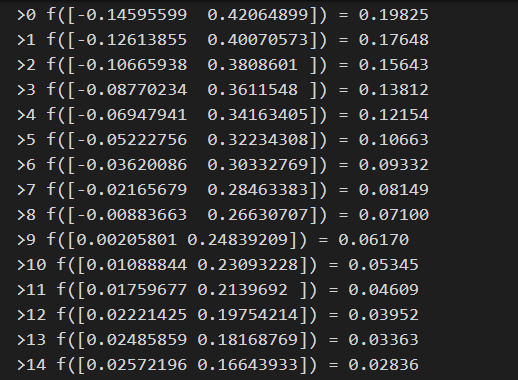
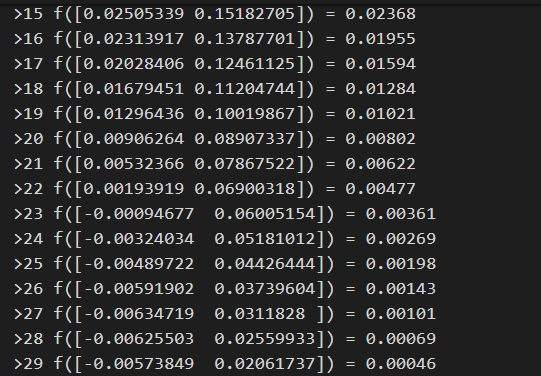
solutions = asarray(solutions)

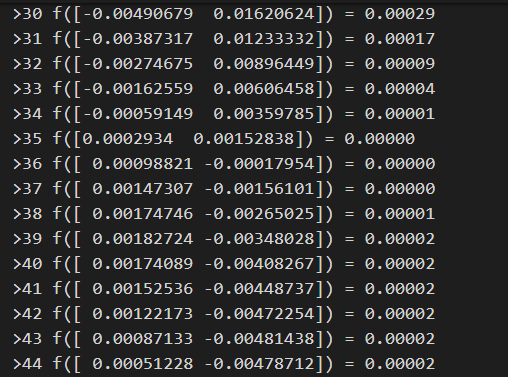
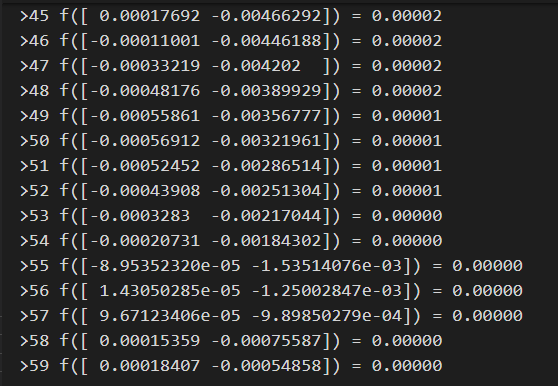
pyplot.plot(solutions[:, 0], solutions[:, 1], '.-', color='g')

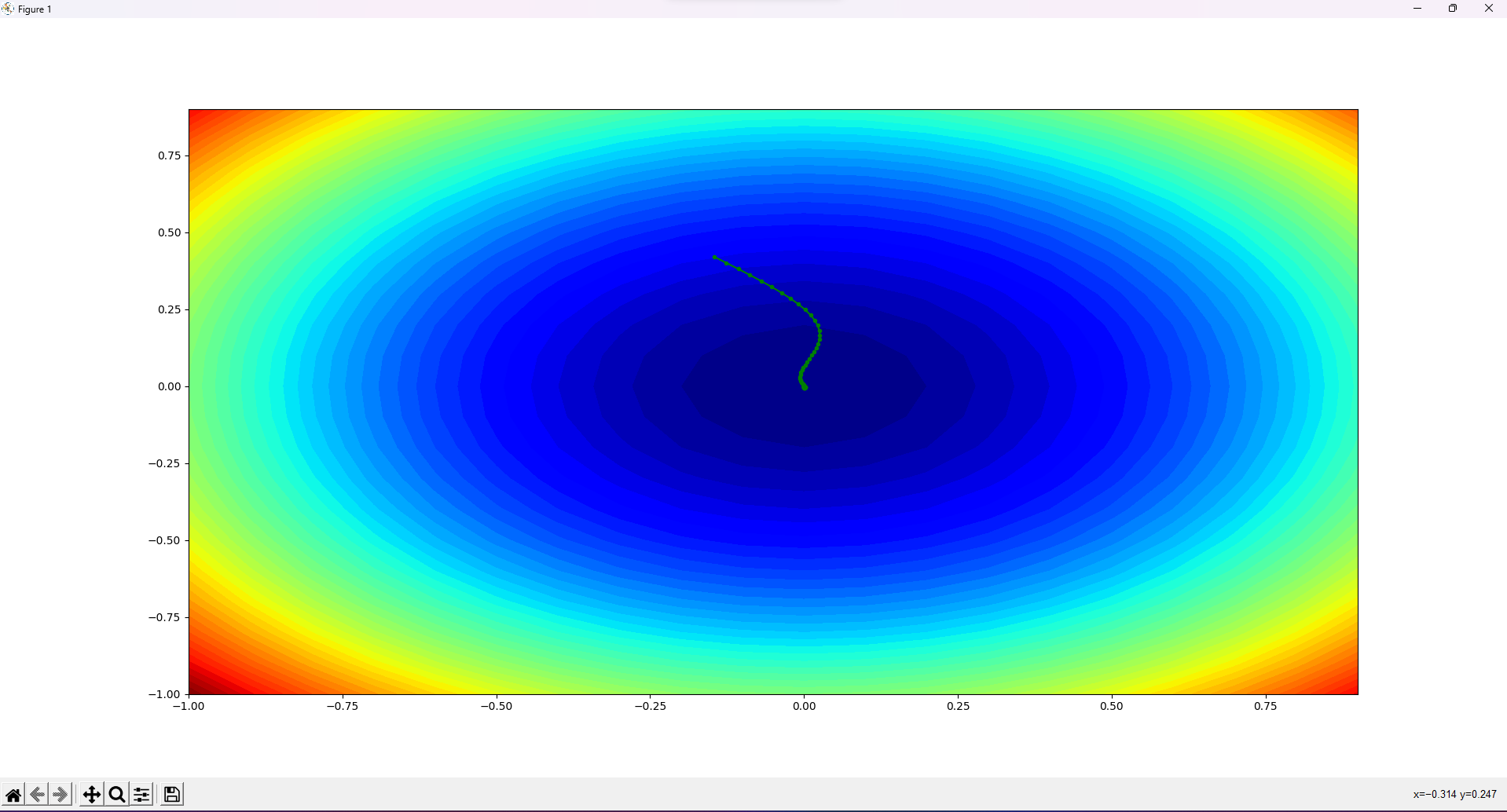
# show the plot

pyplot.show()

**Output:**

**** ****

**** ****

****