Warren Fernandes

TE COMPS B Batch B

8940

Genetic Algorithm: -

Find the smallest x possible for the equation x+1/x

# genetic algorithm search for continuous function optimization from numpy.random import randint from numpy.random import rand

# objective function

def objective(x): return x[0]+1/x[0]

# decode bitstring to numbers def decode(bounds, n\_bits, bitstring):

decoded = list() largest = 2\*\*n\_bits for i in range(len(bounds)): # extract the substring start, end = i \* n\_bits, (i \* n\_bits)+n\_bits substring = bitstring[start:end] # convert bitstring to a string of chars chars = ''.join([str(s) for s in substring])

# convert string to integer integer = int(chars, 2) # scale integer to desired range value = bounds[i][0] + (integer/largest) \* (bounds[i][1] - bounds[i][0])

# store decoded.append(value) return decoded

# tournament selection def selection(pop, scores, k=3): # first random selection selection\_ix = randint(len(pop)) for ix in randint(0, len(pop), k-1):

# check if better (e.g. perform a tournament) if scores[ix] < scores[selection\_ix]:

selection\_ix = ix return pop[selection\_ix]

# crossover two parents to create two children def crossover(p1, p2, r\_cross):

# children are copies of parents by default c1, c2 = p1.copy(), p2.copy() # check for recombination if rand() < r\_cross:

# select crossover point that is not on the end of the string pt = randint(1, len(p1)-2)

# perform crossover c1 = p1[:pt] + p2[pt:] c2 = p2[:pt] + p1[pt:]

return [c1, c2]

# mutation operator def mutation(bitstring, r\_mut): for i in range(len(bitstring)): # check for a mutation if rand() < r\_mut: # flip the bit bitstring[i] = 1 - bitstring[i]

# genetic algorithm def genetic\_algorithm(objective, bounds, n\_bits, n\_iter, n\_pop, r\_cross, r\_mut):

# initial population of random bitstring pop = [randint(0, 2, n\_bits\*len(bounds)).tolist() for \_ in range(n\_pop)]

# keep track of best solution best, best\_eval = 0, objective(decode(bounds, n\_bits, pop[0]))

# enumerate generations for gen in range(n\_iter): # decode population decoded = [decode(bounds, n\_bits, p) for p in pop] # evaluate all candidates in the population scores = [objective(d) for d in decoded]

# check for new best solution for i in range(n\_pop): if scores[i] < best\_eval:

best, best\_eval = pop[i], scores[i] print(">%d, new best f(%s) = %f" % (gen, decoded[i], scores[i]))

# select parents selected = [selection(pop, scores) for \_ in range(n\_pop)] # create the next generation children = list() for i in range(0, n\_pop, 2): # get selected parents in pairs p1, p2 = selected[i], selected[i+1] # crossover and mutation for c in crossover(p1, p2, r\_cross):

# mutation mutation(c, r\_mut) # store for next generation children.append(c) # replace population pop = children return [best, best\_eval]

# define range for input bounds = [[1.0, 100.0], [1.0, 100.0]] # define the total iterations n\_iter = 100 # bits per variable n\_bits = 16

# define the population size n\_pop = 100 # crossover rate r\_cross = 0.25 # mutation rate r\_mut = 1.0 / (float(n\_bits) \* len(bounds)) # perform the genetic algorithm search

best, score = genetic\_algorithm(objective, bounds, n\_bits, n\_iter, n\_pop, r\_cross, r\_mut) print('Done!') decoded = decode(bounds, n\_bits, best) print('f(%s) = %f' % (decoded, score))

OUTPUT:-

# >0, new best f([24.008255004882812, 67.03675842285156]) = 24.049907

# >0, new best f([9.967041015625, 13.468658447265625]) = 10.067372

# >0, new best f([5.3369903564453125, 27.59295654296875]) = 5.524362

# >0, new best f([2.900360107421875, 45.71586608886719]) = 3.245145

# >1, new best f([2.59521484375, 75.99775695800781]) = 2.980539

# >1, new best f([1.510589599609375, 61.86137390136719]) = 2.172583

# >4, new best f([1.4728240966796875, 89.80180358886719]) = 2.151792 # >5, new best f([1.123870849609375, 58.76762390136719]) = 2.013653

# >7, new best f([1.1102752685546875, 83.13090515136719]) = 2.010953

# >7, new best f([1.0861053466796875, 21.739303588867188]) = 2.006826

# >9, new best f([1.027191162109375, 68.05038452148438]) = 2.000720

# >10, new best f([1.009063720703125, 93.37895202636719]) = 2.000081

# >13, new best f([1.00604248046875, 68.62895202636719]) = 2.000036

# >15, new best f([1.0015106201171875, 21.450775146484375]) = 2.000002

# >15, new best f([1.0, 46.10258483886719]) = 2.000000

# Done!

# f([1.0, 46.10258483886719]) = 2.000000