ICA for send

January 7, 2022

[1]:
```
import numpy as np
import math
```

[2]:
```
num_countries = 100

num_imp = 10

lbound = 0

ubound = 10

beta = 2

teta = np.random.uniform(low=-math.pi/4, high=math.pi/4)

zeta = 0.1

num_of_goals = 2

algorithem_iteration = 10
```

[3]:
```
def func(w):
    # G1 function from the original ICA paper
    return ((w[0] * math.sin(4*w[0])) + (1.1 * w[1] * math.sin(2*w[1])))
```

[13]:
```
def ICA():

    end_results=[[] for i in range(num_of_goals)]

    for _ in range(algorithem_iteration):
        #a

        colony = np.random.randint(lbound, high=ubound, size=(num_countries,␣
↪2+1)).astype("float")

        for i in range(len(colony)):
            colony[i][2] = func(colony[i])
```

```
        sorted_colony = colony[colony[:, 2].argsort()]


        imps=[]

        for i in range(num_imp):

            imps.append(sorted_colony[i])
            imps[i] = np.insert(imps[i],3,i)

            sorted_colony = np.delete(sorted_colony, len(sorted_colony)-1, 0)␣
↪#sorted_colony[len(sorted_colony)-1-i]

        C=[]
        for i in range(len(imps)):
            C.append(imps[i][2] + imps[0][2])

        P = []
        for i in range(len(imps)):
            P.append(C[i]/(sum(C)))

        num_col = []
        for i in range(len(imps)):
            num_col.append(round(P[i] * (num_countries-len(imps)-1)))

        # making sure right number of colonies are going to be given to the␣
↪empires----------------------

        while sum(num_col) > (num_countries-len(imps)-1) :
            num_col[len(num_col)-1] -= 1

        while sum(num_col) < (num_countries-len(imps)-1) :
            num_col[0] += 1

        # give the colonies to the␣
↪empires-----------------------------------------------------------------

        imps_belonging = []

        np.random.shuffle(sorted_colony)

        w=0

        dd = sorted_colony

        while w < (num_countries-len(imps)-1) :
```

```
            for i in range(len(imps)):

                for j in range(num_col[i]):

                    ll = np.insert(dd[w],3,i)
                    imps_belonging.append( ll )

                    w += 1
        #with the while loop we can say how many mins(goals) to look for␣
↪---------------------------------------
        iterr = 0
        while len(imps) != num_of_goals :

            test_last = imps_belonging
            # getting colonies closer to the␣
↪empires-----------------------------------------------------------------

            www = 0


            for i in range(len(imps)):

                for j in range(num_col[i]):

                    dist = math.sqrt((imps_belonging[www][0]-imps[i][0])**2 +␣
↪(imps_belonging[www][1]-imps[i][1])**2)

                    x = np.random.uniform(low=0, high=beta*dist)

                    imps_belonging[www][0] += x*math.cos(teta)
                    imps_belonging[www][1] += x*math.sin(teta)

                    if imps_belonging[www][0] > 10 :
                        imps_belonging[www][0] = 10
                    elif imps_belonging[www][0] < 0 :
                        imps_belonging[www][0] = 0

                    if imps_belonging[www][1] >10 :
                        imps_belonging[www][1] = 10
                    elif imps_belonging[www][1] < 0 :
                        imps_belonging[www][1] = 0

                    www+=1

            # re-evaluate(calculating fitness␣
↪again)-------------------------------------------------------------
```

```
            for i in range(len(imps_belonging)):
                imps_belonging[i][2] = func(imps_belonging[i])

            er=np.array(imps_belonging)

            imps_belonging = er[er[:, 3].argsort()]

            #replacing the empire with the better␣
↪colony-------------------------------------------------------

            w = 0


            for i in range(len(imps)):

                for j in range(num_col[i]):

                    # find the best(min)
                    if imps_belonging[w][2] < imps[i][2] :
                        cc = imps[i]
                        imps[i] = imps_belonging[w]
                        imps_belonging[w] = cc

                    w+=1
            #mean␣
↪cost---------------------------------------------------------------------------------------

            w = 0

            mean_cost=[]



            for i in range(len(imps)):
                r = 0

                for j in range(num_col[i]):

                    r += imps_belonging[w][2]
                    w+=1

                mean_cost.append(r/num_col[i])

            # Total␣
↪cost---------------------------------------------------------------------------------------

            total_cost = []
```

```
            for i in range(len(imps)):

                total_cost.append(imps[i][2] + zeta * mean_cost[i] )

            # weakest empire (we want to minimize our function␣
↪here)-----------------------------------------------

            check_imp = total_cost.index(max(total_cost))

            # weakest colony of the weakest␣
↪empire-----------------------------------------------------------------

            start = sum(num_col[:check_imp])

            l = imps_belonging[start: start+num_col[check_imp]]

            ll=[]
            for i in range(len(l)):
                ll.append(l[i][2])

            weak_col_index = ll.index(max(ll))

            # Normalized Total␣
↪cost-------------------------------------------------------------------------------------

            normal_total_cost = []

            for i in range(len(total_cost)):

                normal_total_cost.append(total_cost[i]-min(total_cost))

            # calculating PP , R ---> D =␣
↪pp-R-------------------------------------------------------------------

            pp = []

            for i in range(len(normal_total_cost)):

                pp.append(abs(normal_total_cost[i]/sum(normal_total_cost)))


            R = [np.random.uniform(low=0, high=1) for _ in range(len(pp))]

            D = [(pp[a] - R[a]) for a in range(len(pp))]
```

```
            # best empire to take control of the␣
↪colony-------------------------------------------------------------

            best_choice = D.index(min(D))

            #weakest colony ---> give it to the best␣
↪empire-------------------------------------------------------------

            imps_belonging[start: start+num_col[check_imp]][weak_col_index][3]␣
↪= best_choice

            # sorting colonies because this is importatnt in this␣
↪code-----------------------------------------------

            er=np.array(imps_belonging)
            imps_belonging = er[er[:, 3].argsort()]


            # see if there is a dead empire and if ther is --> give the dead␣
↪empire to the strong empire

            past_imps = len(imps)

            for i in range(past_imps):

                count = 0

                for j in range(len(imps_belonging)):


                    if imps_belonging[j][3] == i :

                        count+=1


                if count == 0 :

                    #giving the fallen empire to the most powerfull␣
↪one----------------------------------------------------

                    imps[i][3] = best_choice

                    dead_imp = imps[i]

                    imps = np.delete(imps, i, 0)
```

```
                    imps_belonging = list(imps_belonging)

                    imps_belonging.append(dead_imp)

                    er=np.array(imps_belonging)

                    imps_belonging = er[er[:, 3].argsort()]

        for i in range(num_of_goals):
            end_results[i].append(imps[i][2])
        print("winning imperialists =\n(x, y, function value, number of␣
↪imperialist)\n", imps,)
    for i in range(len(end_results)):
        end_results[i] = sum(end_results[i])/len(end_results[i])
    print("\nMean function value of goal1, goal2, ..., goaln after␣
↪{algorithem_iteration} repeats= ".
↪format(algorithem_iteration=algorithem_iteration), end_results)
```

1 The first min of G1 function : x=9.039, y=8.668, function value = -18.5547

[23]:
```
ICA()
```

```
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  6.           9.         -12.86824392   1.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           8.         -11.45955887   0.        ]
 [  9.           8.         -11.45955887   0.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           5.         -11.91812579   0.        ]
 [  9.           8.         -11.45955887   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   4.        ]]
```

```
winning imperialists =
(x, y, function value, number of imperialist)
 [[  6.           9.         -12.86824392   0.        ]
 [  9.           8.         -11.45955887   3.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.08838317   2.07781271 -10.79926213   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           5.         -11.91812579   0.        ]
 [  9.           5.         -11.91812579   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   3.        ]]

Mean function value of goal1, goal2, …, goaln after 10 repeats=
[-14.632875490864492, -11.983387220287215]
```

[18]:
```
ICA()
```

```
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           5.         -11.91812579   4.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.0178043    8.65688067 -18.51984446   0.        ]
 [  9.           8.         -11.45955887   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  6.           9.         -12.86824392   0.        ]
 [  9.           5.         -11.91812579   3.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  6.           9.         -12.86824392   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   1.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
```

```
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   1.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  6.           9.         -12.86824392   0.        ]
 [  9.07967633   5.07014719 -12.57427032   1.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           6.         -12.46739094   2.        ]
 [  9.           6.         -12.46739094   1.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  6.           9.         -12.86824392   2.        ]]

Mean function value of goal1, goal2, …, goaln after 10 repeats=
[-15.488842377293157, -12.347613235810508]
```

[ ]: