HOMEWORK 1 FOR MODERN OPTIMIZATION METHODS
FALL SEMESTER 2021
DUE DATE: 2021/10/13 09:15

1. **Optimization of a Function using Simulated Annealing.** *(35 points)* Consider the following function:

$$f(X) = 6x_1^2 - 6x_1x_2 + 2x_2^2 - x_1 - 2x_2$$

(a) Find the minimum of the function using simulated annealing. Assume suitable parameters and show detailed calculations for 2 iterations.

$f(x) = 6x_1^2 - 6x_1x_2 + 2x_2^2 - x_1 - 2x_2$

Step1. selecting random points. $\Rightarrow X^{(1)} = \begin{bmatrix}2\\0\end{bmatrix}, X^{(2)} = \begin{bmatrix}5\\10\end{bmatrix}, X^{(3)} = \begin{bmatrix}8\\5\end{bmatrix}, X^{(4)} = \begin{bmatrix}10\\10\end{bmatrix}$

$\Rightarrow f^{(1)} = 22,\ f^{(2)} = 25,\ f^{(3)} = 176,\ f^{(4)} = 170.$

$\Rightarrow T = (22+25+176+170)/4 = 92.5$

$\Rightarrow C = 0.5$ (temperature reduction factor), $X_1 = \begin{bmatrix}4\\5\end{bmatrix}$

Step2. $f_1 = 12$, iteration = 1 ($i=1$)

Step3. randomly select $u_1 = 0.31$ for x_1 in the vicinity of 4

$u_2 = 0.57$ for x_2 in the vicinity of 5

choose range of $x_1 = (-2, 10)$ and $x_2 = (-1, 11) \Rightarrow$ (range of ± 6)

$\Rightarrow$ uniformly distributed random numbers: $r_1 = -2 + 0.31(10-(-2)) = 1.72$

$r_2 = -1 + 0.57(11-(-1)) = 5.84$

$\Rightarrow X_2 = \begin{bmatrix}r_1\\r_2\end{bmatrix} = \begin{bmatrix}1.72\\5.84\end{bmatrix} \Rightarrow f_2 = 12.2928$

$\Rightarrow \Delta f = f_2 - f_1 = 12.2928 - 12 = 0.2928 > 0$

Step 4. Metropolis criterion $\Rightarrow$ random number in the range $(0,1) = r = 0.8$

$\Rightarrow P(X_2) = e^{-\Delta f/KT} = e^{-0.2928/92.5} = 0.9968 > 0.8 \Rightarrow$ accept

Step3. iteration = 2, ($i=2$), $X_2 = \begin{bmatrix}1.72\\5.84\end{bmatrix}$

$\Rightarrow$ range of $\pm 6 \rightarrow x_1 = (-6+1.72, 6+1.72) = (-4.28, 7.72)$

$x_2 = (-6+5.84, 6+5.84) = (-0.16, 11.84)$

$\Rightarrow u_1 = 0.92,\ u_2 = 0.73 \Rightarrow r_1 = -4.28 + 0.92(7.72-(-4.28)) = 6.76$

$r_2 = -0.16 + 0.73(11.84-(-0.16)) = 8.6$

$\Rightarrow X_3 = \begin{bmatrix}6.76\\8.6\end{bmatrix} \rightarrow f_3 = 49.3296 \Rightarrow \Delta f = 49.3296 - 12.2928 = 37.0368 > 0$

Step 4. Metropolis criterion $\Rightarrow$ random number in range $(0,1) = r = 0.5$

$\Rightarrow P(X_3) = e^{-37.0368/92.5} = 0.67 > 0.5 \Rightarrow$ accept.

$\Rightarrow$ cycle of iteration completed $\rightarrow i=3,\ i>2 \Rightarrow$ step 5.

Step5. $T = 0.5 \times 92.5 = 46.25$

reset $i=1$ and go to step 3,

Generate a new design point in the vicinity of X_3 and continue procedure until the T reduced to a small value (convergence). ☆

(b) Write a program of hill climbing and run it to find the minimum solution of the function $f(X)$. Use the results in (a) for initialization.

```
from __future__  import print_function
import math
import random
from matplotlib import pyplot
import numpy as np

def function(x):
    r = 6*x[0]*x[0] - 6*x[0]*x[1] + 2*x[1]*x[1] - x[0] -2*x[1]
    return r

#initial = [random.uniform(-40,40), random.uniform(-40,40)]
initial = [6.76, 8.6]
#initial = [20,20]

# ================== Hill climbing ================== #

def hill_climbing(N, variables, x):
    result = []
    walk_num = 1
    while(len(result)<10):
        u = [random.uniform(-1,1) for i in range(variables)]
        u1 = [u[i]/math.sqrt(sum([u[i]**2 for i in range(variables)])) for i in range(variables)]
        x1 = [x[i] + u1[i] for i in range(variables)]
        if(function(x1) < function(x)): # Find Better Solution
            x = x1 # 下次從最佳解附近找candidate
            result.append(function(x))
        walk_num += 1
        #print(walk_num)

    print("hill_climbing最終最佳點:",x)
    print("hill_climbing最終最佳解:",function(x))

    return result

hill_climbing = hill_climbing(10, 2, initial)
```

```
hill_climbing最終最佳點: [1.0829711103890969, 2.345675527492436]
hill_climbing最終最佳解: -2.9747692329218562
```

(c) Write a program of random walk and run it to find the minimum solution of the function $f(X)$. Use the results in (a) for initialization.

```
# ================== Random Walk ==================
def random_walk(N, variables, x):
    result = []
    pbest = function(x)
    pbest_ = x
    walk_num = 1
    while(len(result)<10):
        u = [random.uniform(-1,1) for i in range(variables)] # 隨機變數
        u1 = [u[i]/math.sqrt(sum([u[i]**2 for i in range(variables)])) for i in range(variables)]
        x1 = [x[i] + u1[i] for i in range(variables)]
        if(function(x1) < pbest): # Find Better Solution
            pbest = function(x1) # 存取歷史最佳解
            pbest_ = x1 # 存取歷史最佳點
            result.append(function(x1))
            print(len(result))
        x = x1 # 下次從目前所在位置繼續搜尋candidate
        walk_num += 1

    print("random_walk最終最佳點:",pbest_)
    print("random_walk最終最佳解:",pbest)

    return result


random_walk = random_walk(10, 2, initial)
```

```
random_walk最終最佳點: [2.28429608569083, 3.2410501103072553]
random_walk最終最佳解: -0.870641511661491
```

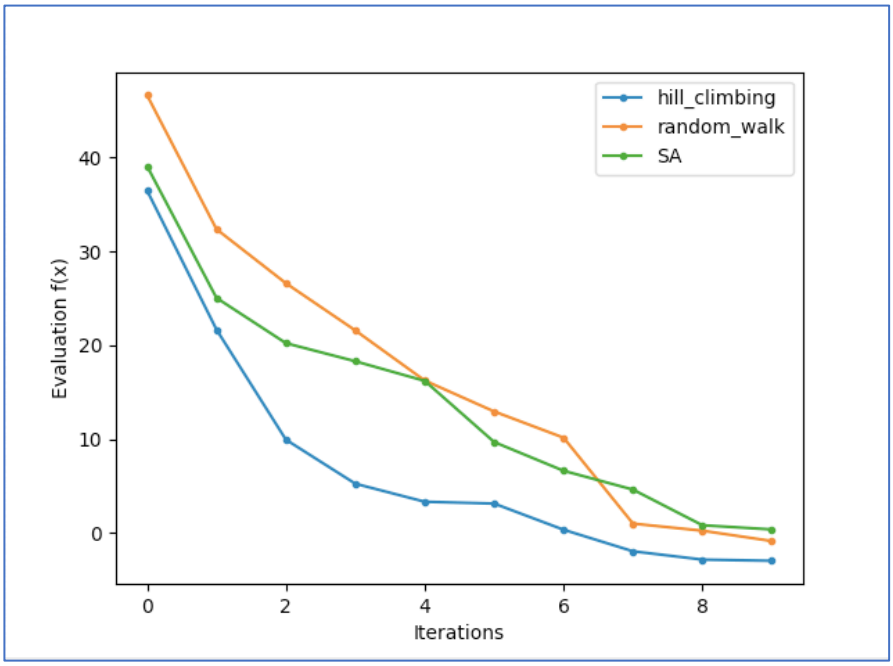
(d) Write a program of Simulation Annealing and run it to find the minimum solution of the function $f(X)$. Use the results in (a) for initialization.

```
# ================= SA =================
def SA(eta, k, t, initial):
    result = []
    x_old = initial
    x_best = function(x_old)
    best = x_old

    #times = 0
    #while times < 100:
    while t>=0.2:
        value_old = function(x_old)
        x_new = [x_old[0] + random.uniform(-1,1), x_old[1] + random.uniform(-1,1)]
        value_new = function(x_new)
        res = value_new-value_old
        if res<0 or np.exp(-res/(k*t))>np.random.rand():
            x_old = [x_new[0], x_new[1]]
            if value_new < x_best:
                x_best = function(x_new)
                best = x_new
                result.append(function(x_new))
        t = t*eta
        #times += 1
        if len(result) >= 10:
            break

    print("SA最終最佳點: ",best[0], " " ,best[1])
    print("SA最終最佳解：",x_best)

    return result

SA = SA(0.95, 1, 1000, initial)
```

```
SA最終最佳點:  2.4113358984744084   5.062968483961428
SA最終最佳解：  0.36616581507993295
```

(e) Compare three algorithms via the best $f(x)$ versus FE plot.

```
# line plot of best scores
pyplot.plot(hill_climbing, '.-', label = "hill_climbing")
pyplot.plot(random_walk, '.-', label = "random_walk")
pyplot.plot(SA, '.-', label = "SA")
pyplot.legend(loc='upper right')
pyplot.xlabel('Times')
pyplot.ylabel('Evaluation f(x)')
pyplot.show()
```



2. **Optimization of Travel Routes for South Korea Cities.** *(65 points)* Assume you want to organize a travel trip to visit cities in South Korea. It is obvious that you want to minimize the distance travelled. You arrive at and leave from South Korea via Incheon. Here is the 15 cities you plan to visit. Incheon, Seoul, Busan, Daegu, Daejeon, Gwangju, Suwon-si, Ulsan, Jeonju, Cheongju-si, Changwon, Jeju-si, Chuncheon, Hongsung, Muan.

(a) Create the distance of location table. In this problem, the table will be symmetric over the diagonal because the distance between City A and City B is fixed without direction difference.

	Incheon	Seoul	Busan	Daegu	Daejeon	Gwangju	Suwon-si	Ulsan	Jeonju	Cheongju-si	Changwon	Jeju-si	Chuncheon	Hongsung	Muan
Incheon	0	27	335	244	141	257	33	316	186	115	304	439	102	95	275
Seoul	27	0	330	237	144	268	31	307	195	113	301	453	75	111	290
Busan	335	330	0	95	199	193	304	54	189	221	35	291	330	271	233
Daegu	244	237	95	0	117	171	212	75	130	130	72	324	236	191	215
Daejeon	141	144	199	117	0	137	114	192	61	36	167	323	175	74	171
Gwangju	257	268	193	171	137	0	238	222	77	173	161	186	311	162	44
Suwon-si	33	31	304	212	114	238	0	284	164	84	274	423	91	83	260
Ulsan	316	307	54	75	192	222	284	0	198	205	67	341	296	266	265
Jeonju	186	195	189	130	61	77	164	198	0	96	154	263	234	97	111
Cheongju-si	115	113	221	130	36	173	84	205	96	0	190	359	139	74	205
Changwon	304	301	35	72	167	161	274	67	154	190	0	275	306	237	202
Jeju-si	439	453	291	324	323	186	423	341	263	359	275	0	498	344	165
Chuncheon	102	75	330	236	175	311	91	296	234	139	306	498	0	170	340
Hongsung	95	111	271	191	74	162	83	266	97	74	237	344	170	0	180
Muan	275	290	233	215	171	44	260	265	111	205	202	165	340	180	0

(b) Calculate how many evaluation of objective function required if one attempts the exhaustive enumeration.

Ans: 14! / 2

(c) Run a random walk to find the optimal path that passes through these 15 cities with minimum distance. Report the optimal path obtained from the random walk after 100 iterations. Set appropriate values for parameters if needed.

```
Random Walk:
最佳路線： [0, 13, 12, 6, 10, 2, 11, 8, 14, 5, 7, 3, 4, 1, 9]
最佳路線長度： 2160
```

```python
# ========================== random_walk ==========================

def randomSolution_2(tsp):
    cities = [num for num in range(1,15)]
    solution = []
    solution.append(0)
    for i in range(len(tsp)-1):
        randomCity = cities[random.randint(0, len(cities) - 1)]
        solution.append(randomCity)
        cities.remove(randomCity)
    return solution

def routeLength_2(tsp, solution):
    routeLength = 0
    for i in range(len(solution)):
        routeLength += tsp[solution[i - 1]][solution[i]]
    return routeLength

def getNeighbours_2(solution):
    neighbours = []
    for i in range(1,len(solution)):
        for j in range(i + 1, len(solution)):
            neighbour = solution.copy()
            neighbour[i] = solution[j]
            neighbour[j] = solution[i]
            neighbours.append(neighbour)
    return neighbours


def random_walk(tsp):
    result = []
    time = 1

    currentSolution = randomSolution_2(tsp)
    currentRouteLength = routeLength_2(tsp, currentSolution)
    neighbours = getNeighbours_2(currentSolution)

    random_ = random.randrange(0, len(neighbours))
    neighbor = neighbours[random_]
    RouteLength = routeLength_2(tsp, neighbor)

    bestNeighbour = neighbor
    bestNeighbourRouteLength = RouteLength
    BEST = bestNeighbour # 初始化最佳解
    BEST_LENGTH = bestNeighbourRouteLength # 初始化最佳解長度

    while time<=100:
        currentSolution = bestNeighbour
        currentRouteLength = bestNeighbourRouteLength

        if (currentRouteLength < BEST_LENGTH):
            BEST = currentSolution
            BEST_LENGTH = currentRouteLength

        result.append(BEST_LENGTH)
        neighbours = getNeighbours_2(currentSolution)
        random_ = random.randrange(0, len(neighbours))
        bestNeighbour = neighbours[random_]
        bestNeighbourRouteLength = routeLength_2(tsp, bestNeighbour)
        time += 1
    return BEST, BEST_LENGTH, result
```

(a) Run a hill climbing to find the optimal path that passes through these 15 cities with minimum distance. Record the distances of the best paths in all 100 iterations. Set appropriate values for parameters if needed.

```
Hill Climbing:
最佳路線： [0, 13, 8, 4, 7, 2, 10, 3, 5, 14, 11, 9, 12, 6, 1]
最佳路線長度： 1633
```

```python
# ========================= hill_climbing =========================

def randomSolution(tsp):
    #cities = list(range(len(tsp)))
    cities = [num for num in range(1,15)]
    solution = []
    solution.append(0)
    for i in range(len(tsp)-1):
        randomCity = cities[random.randint(0, len(cities) - 1)]
        solution.append(randomCity)
        cities.remove(randomCity)
    return solution

def routeLength(tsp, solution):
    routeLength = 0
    for i in range(len(solution)):
        routeLength += tsp[solution[i - 1]][solution[i]]
    return routeLength

def getNeighbours(solution):
    neighbours = []
    for i in range(1,len(solution)):
        for j in range(i + 1, len(solution)):
            neighbour = solution.copy()
            neighbour[i] = solution[j]
            neighbour[j] = solution[i]
            neighbours.append(neighbour)
    return neighbours


def getNeighbours(solution):
    neighbours = []
    for i in range(1,len(solution)):
        for j in range(i + 1, len(solution)):
            neighbour = solution.copy()
            neighbour[i] = solution[j]
            neighbour[j] = solution[i]
            neighbours.append(neighbour)
    return neighbours

def getBestNeighbour(tsp, neighbours, time):
    bestRouteLength = routeLength(tsp, neighbours[0])
    bestNeighbour = neighbours[0]
    for neighbour in neighbours:
        currentRouteLength = routeLength(tsp, neighbour)
        if currentRouteLength < bestRouteLength:
            bestRouteLength = currentRouteLength
            bestNeighbour = neighbour
        time+=1
        #print(time)
    return bestNeighbour, bestRouteLength, time


def hillClimbing(tsp):
    result = []
    time = 1

    currentSolution = randomSolution(tsp)
    currentRouteLength = routeLength(tsp, currentSolution)
    neighbours = getNeighbours(currentSolution)

    bestNeighbour = neighbours[0]
    bestNeighbourRouteLength = routeLength(tsp, bestNeighbour)

    while time<=100:
        neighbours = getNeighbours(currentSolution)

        random_ = random.randrange(0, len(neighbours))
        neighbor = neighbours[random_]
        RouteLength = routeLength(tsp, neighbor)
        if RouteLength <= bestNeighbourRouteLength:
            bestNeighbourRouteLength = RouteLength
            currentSolution = neighbor # 從現在位置繼續找

        result.append(bestNeighbourRouteLength)
        time += 1
        #print("time：",time)

    return currentSolution, bestNeighbourRouteLength, result
```

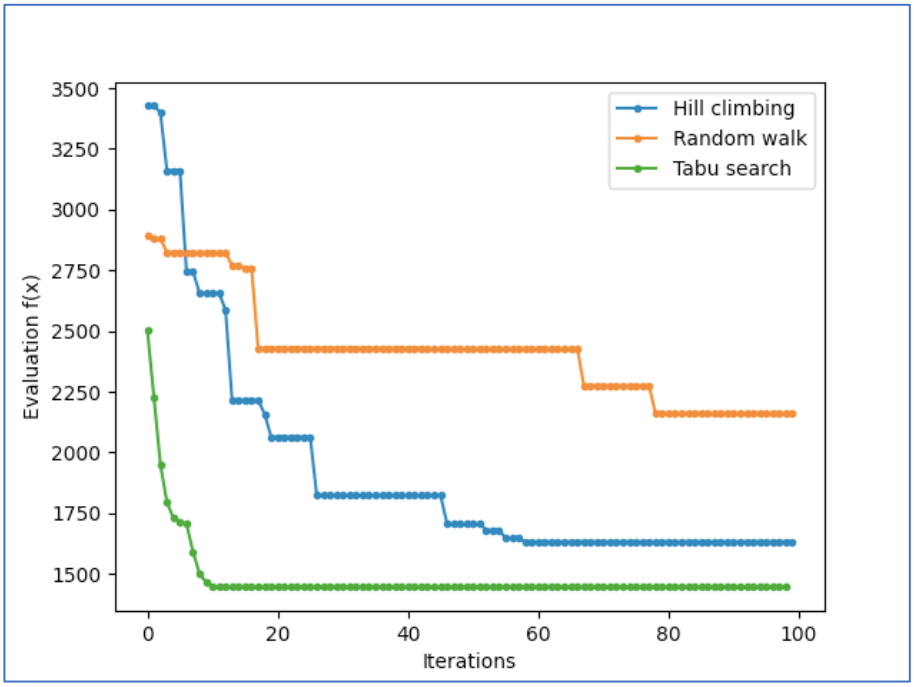
(d) Write a program of Tabu Search for traveling salesman problem.

(e) Set maximum iteration to be 100, maximum length of tabu list to be 10, aspiration criterion as expectation improvement, and we swap during the move. Use a random point as the starting point, run tabu search to find the optimal path that passes through these 15 cities (and return to Incheon at the end) with minimum distance.

```
Tabu Search:
最佳路線： [0, 1, 12, 6, 9, 4, 13, 8, 11, 2, 7, 10, 3, 5, 14]
最佳路線長度： 1446
Tabu List： [[12, 13], [10, 11], [7, 13], [10, 13], [5, 6], [2, 3]]
```

```python
# ========================== tabu_search ==========================

def path_length(path, distance):
    cities_pairs = zip(path, path[1:])
    consecutive_distances = [(distance[a][b]) for (a, b) in cities_pairs]
    return round(sum(consecutive_distances), 2)


def tabu_search(distance):
    ilosc = 15 # 城市個數
    iter_ = 1

    pbest = [n for n in range(ilosc)]
    pbest_x = pbest
    pbest_y = path_length(pbest_x, distance)
    pcur_x = pbest_x
    pcur_y = pbest_y
    tabu = []
    moveb = []
    score = []
    #for n in range(1,iter_):
    while iter_<=100:
        pnew = []
        for i in range(1, len(pbest_x)-1):
            for j in range(2, len(pbest_x)-1):
                if (i != j) and (i < j):
                    acopy = pbest_x
                    acopy[i], acopy[j] = acopy[j], acopy[i]
                    acopy_ = acopy
                    length = path_length( acopy_,distance )
                    move = [i,j]

                    if ( ( (moveb not in tabu) ) and
                         ( ( length<path_length(pnew,distance) ) or ( pnew==[] ) )
                         or (length<=pbest_y) ):
                        if pnew != []:
                            pnew = []
                            for i in acopy_:
                                pnew.append(i)
                        else:
                            for i in acopy_:
                                pnew.append(i)
                        #pnew = acopy_
                        moveb = move

        pcur_x = pnew
        if pcur_x!=[]:
            if path_length(pcur_x,distance) <= path_length(pbest_x,distance):
                pbest_x = pcur_x
                pbest_y = path_length(pcur_x,distance)
                score.append(pbest_y)
            if moveb not in tabu:
                tabu.append(moveb)
            if len(tabu) >= 10:
                tabu.pop(0)

            iter_ += 1
            if iter_ >= 100:
                break
            #else:
                #score.append(pbest_y)

    return pbest_x, tabu, path_length(pbest_x,distance), score
```

(f) Create the best distance vs iterations plot by plotting the results of Hill Climbing, Random Walk and Tabu Search on the same plot. It should be similar to best f(x) vs FEs plot in the lecture note 02-3.




```
Hill Climbing:
最佳路線： [0, 13, 8, 4, 7, 2, 10, 3, 5, 14, 11, 9, 12, 6, 1]
最佳路線長度： 1633
Random Walk:
最佳路線： [0, 13, 12, 6, 10, 2, 11, 8, 14, 5, 7, 3, 4, 1, 9]
最佳路線長度： 2160
Tabu Search:
最佳路線： [0, 1, 12, 6, 9, 4, 13, 8, 11, 2, 7, 10, 3, 5, 14]
最佳路線長度： 1446
Tabu List： [[12, 13], [10, 11], [7, 13], [10, 13], [5, 6], [2, 3]]
```

(g) Compare these three algorithms and comments on their strength and weakness in this problem.

(1) Hill Climbing:

- ✓ Strength: 對於區域最佳解的收斂速度非常快。
- ✓ Weakness: 可能陷入局部最優解而找不到 Global 的最佳解。

(2) Random Walk:

- ✓ Strength: 到達了區域最佳解後會繼續搜索，有機會達到全域最佳解。
- ✓ Weakness: 較 Hill Climbing 為 Global，但找到的也不一定為 Global 最佳解。

(3) Tabu Search:

- ✓ Strength: 可以將曾經走過的路徑記起來，減少計算量，收斂速度較快。
- ✓ Weakness: 可能只搜索到區域最佳解，且演算法的效能對於參數及初始值的設定非常敏感。

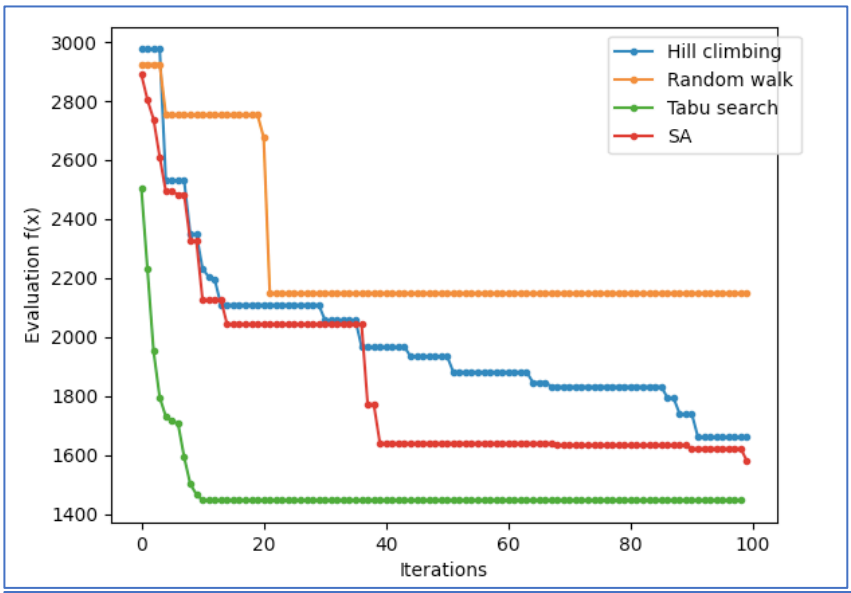
3. **More on Optimization of Travel Routes for South Korea Cities.** *(Bonus 10 points)* We try to evaluate the performance of hill climbing, random walk, simulated annealing and tabu search in the optimization of travel route problem.

   (a) In fact, simulated annealing can also be used for finding this optimal path. Run simulated annealing to find the optimal path that passes through these 15 cities with minimum distance. Set appropriate values for parameters if needed.

```
Simulated Annealing:
最佳路線： [ 0  1  6  9  4  3 10  7  2  8  5 14 11 13 12]
最佳路線長度： 1579
```

```
# ========================== SA ==========================

def SA(distmat):
    num = 15 #城市數量
    distmat = np.asarray(distmat)
    solutionnew = np.arange(num)
    solutioncurrent = solutionnew.copy()
    valuecurrent =99000
    solutionbest = solutionnew.copy()
    valuebest = 99000
    iter_ = 100
    result = []
    time = 0
    solutions = []
    while time < iter_:
        loc1 = np.int(np.ceil(np.random.rand()*(num-1)))
        loc2 = np.int(np.ceil(np.random.rand()*(num-1)))
        if loc1 != loc2:
            solutionnew[loc1],solutionnew[loc2] = solutionnew[loc2],solutionnew[loc1]

        valuenew = 0
        for i in range(num-1):
            valuenew += distmat[solutionnew[i]][solutionnew[i+1]]
        valuenew += distmat[solutionnew[0]][solutionnew[14]]
        if valuenew<valuecurrent:
            valuecurrent = valuenew
            solutioncurrent = solutionnew.copy()
            if valuenew < valuebest:
                valuebest = valuenew
                solutionbest = solutionnew.copy()
        else:
            if np.random.rand() < np.exp(-(valuenew-valuecurrent)/time):
                valuecurrent = valuenew
                solutioncurrent = solutionnew.copy()

            else:
                solutionnew = solutioncurrent.copy()

        solutions.append(solutionnew)
        time += 1
        result.append(valuebest)
    best_length = min(result)
    best_ = solutions[result.index(best_length)]

    return best_length, best_, result
```

(b) Create the best distance vs iterations plot by plotting the results of Hill Climbing, Random Walk, Tabu Search and Simulated Annealing on the same plot. It should be similar to best f(x) vs FEs plot in the lecture note 02-3.




```
Hill Climbing:
最佳路線： [0, 1, 12, 6, 3, 7, 2, 10, 11, 14, 4, 8, 5, 9, 13]
最佳路線長度： 1660
Random Walk:
最佳路線： [0, 1, 7, 3, 5, 14, 2, 10, 4, 11, 8, 9, 12, 6, 13]
最佳路線長度： 2149
Tabu Search:
最佳路線： [0, 1, 12, 6, 9, 4, 13, 8, 11, 2, 7, 10, 3, 5, 14]
最佳路線長度： 1579
Tabu List： [[12, 13], [10, 11], [7, 13], [10, 13], [5, 6], [2, 3]]
Simulated Annealing:
最佳路線： [ 0  1  6  9  4  3 10  7  2  8  5 14 11 13 12]
最佳路線長度： 1579
```

(c) Compare these four algorithms and comments on their strength and weakness in this problem.

(1) Hill Climbing:

- ✓ Strength: 對於區域最佳解的收斂速度非常快。
- ✓ Weakness: 可能陷入局部最優解而找不到 Global 的最佳解。

(2) Random Walk:

- ✓ Strength: 到達了區域最佳解後會繼續搜索，有機會達到全域最佳解。
- ✓ Weakness: 較 Hill Climbing 為 Global，但找到的也不一定為 Global 最佳解。

(3) Tabu Search:

- ✓ Strength: 可以將曾經走過的路徑記起來，減少計算量，收斂速度較快。
- ✓ Weakness: 可能只搜索到區域最佳解，且演算法的效能對於參數及初始值的設定非常敏感。

(4) Simulated Annealing:

- ✓ Strength: 防止陷入區域性最優，模擬退火演算法以一定概率接受比當前解差的解，接受差解的概率隨著迭代次數的增加而下降。而在參數設置得宜的情況下，能夠較快找到最佳解。
- ✓ Weakness: 收斂速度較慢，初始值與參數的定義對演算法之效能影響很大。