# HW2 Modern Optimization Methods

## B06702064 會計五 林聖硯

**Code Usage**: Please directly run all the cells in `b06702064_HW2.ipynb` , then you can reproduce all my algorithms' results and FE plot.

**Programming Language**: Python (with jupyter kernel)

## Problem1

### (a)

**Step 1: Initialization**

- N = 4
- $x_{1,0} = -2$, $x_{2,0} = 0$, $x_{3,0} = 1$, $x_{4,0} = 3$
- $c_1 = c_2 = w = 1$

Compute objective value

$$f(x_{1,0}) = -53,\ f(x_{2,0}) = -5,\ f(x_{3,0}) = 19,\ f(x_{4,0}) = -53$$

Update local and global maximum

$$P_1 = -2,\ P_2 = 0,\ P_3 = 1,\ P_4 = 3,\ P_g = 1$$

**Step2:** $i = 1$
Randomly generate two values, $r_1 = 0.3412,\ r_2 = 0.5731$

Compute velocity of each particle:

$$v_{1,1} = 0 + 0.3412 \times (-2 + 2) + 0.5731 \times (1 + 2) = 1.7193$$

$$v_{2,1} = 0 + 0.3412 \times (0 + 0) + 0.5731 \times (1 - 0) = 0.5731$$

$$v_{3,1} = 0 + 0.3412 \times (1 - 1) + 0.5731 \times (1 - 1) = 0$$

$$v_{4,1} = 0 + 0.3412 \times (3 - 3) + 0.5731 \times (1 - 3) = -1.1462$$

Update position:

$$x_{1,1} = -0.2807,\ x_{2,1} = 0.5731,\ x_{3,1} = 1,\ x_{4,1} = 1.8538$$

Compute objective value:

$$f(x_{1,1}) = -10.7228,\ f(x_{2,1}) = 7.3413,\ f(x_{3,1}) = 19,\ f(x_{4,1}) = 42.0361$$

Update local and global maximum

$$P_1 = -0.2807,\ P_2 = 0.5731,\ P_3 = 1,\ P_4 = 1.8538,\ P_g = 1.8538$$

**Step2:** $i = 2$
Randomly generate two values, $r_1 = 0.4728,\ r_2 = 0.2135$

Compute velocity of each particle:

$$v_{1,2} = 1.7193 + 0.4728 \times (-0.2807 + 0.2807) + 0.2135 \times (1.8538 + 0.2807) = 2.1750$$

$$v_{2,2} = 0.5731 + 0.4728 \times (0.5731 - 0.5731) + 0.2135 \times (1.8538 - 0.5731) = 0.8465$$

$$v_{3,2} = 0 + 0.4728 \times (1 - 1) + 0.2135 \times (1.8538 - 1) = 0.1823$$

$$v_{4,2} = -1.1462 + 0.4728 \times (1.8538 + 1.8538) + 0.2135 \times (1.8538 - 1.8538) = -1.1462$$

Update position:

$$x_{1,2} = 1.8943,\ x_{2,2} = 1.4196,\ x_{3,2} = 1.1823,\ x_{4,2} = 0.7076$$

Compute objective value:

$$f(x_{1,2}) = 42.1815,\ f(x_{2,2}) = 31.9309,\ f(x_{3,2}) = 24.5992,\ f(x_{4,2}) = 10.7461$$

Update local and global maximum

$$P_1 = 1.8943,\ P_2 = 1.4196,\ P_3 = 1.1823,\ P_4 = 1.8538,\ P_g = 1.8943$$

## (b)

Reference code:

```
class ParticleSwarmOptimization:
    def __init__(self, max_iter, num_particles, x_upper, x_lower):
        self.max_iter = max_iter
        self.num_particles = num_particles
        self.c1 = 1
        self.c2 = 1
        self.w = 1
        self.x_upper = x_upper
        self.x_lower = x_lower
        self.global_best_obj = 0
        self.global_best_obj = 0

    def compute_objective_function_value(self, x):
        value = - x ** 5 + 5 * (x ** 3) + 20 * x - 5
        return value

    def initialize(self):
        x = [-2, 0, 1, 3]
        v = [0 for i in range(self.num_particles)]
        return x, v

    def generate_two_rv(self):
        r1 = np.random.rand()
        r2 = np.random.rand()
        return r1, r2

    def compute_velocity(self, pos, v, local_best_pos):
        r1, r2 = self.generate_two_rv()
        v = self.w * v + self.c1 * r1 * (local_best_pos - pos) + self.c2 * r2 * (self.global_best_pos - pos)
        return v

    def return_max(self, x, x_next):
        return max(x, x_next)
```

```python
def update_local_best(self, x_next_pos, x_next_obj, x_pos, x_obj):
    local_best_pos = []
    local_best_obj = []
    global_best_pos = None
    for idx in range(self.num_particles):
        if (x_next_obj[idx] >= x_obj[idx]):
            local_best_pos.append(x_next_pos[idx])
            local_best_obj.append(x_next_obj[idx])
        else:
            local_best_pos.append(x_pos[idx])
            local_best_obj.append(x_obj[idx])
    return local_best_pos, local_best_obj

def update_global_best(self, local_best_pos, local_best_obj):
    max_local_obj = max(local_best_obj)
    max_local_idx = local_best_obj.index(max_local_obj)
    if max_local_obj >= self.global_best_obj:
        self.global_best_pos = local_best_pos[max_local_idx]
        self.global_best_obj = local_best_obj[max_local_idx]

def bound_value(self, x):
    x = max(self.x_lower, x)
    x = min(self.x_upper, x)
    return x
```

```python
def solve(self):
    #initialization
    pos, v = self.initialize()
    obj = list(map(self.compute_objective_function_value, pos))
    local_best_pos = pos
    local_best_obj = obj

    self.global_best_obj = max(local_best_obj)
    idx_best_pos = local_best_obj.index(self.global_best_obj)
    self.global_best_pos = local_best_pos[idx_best_pos]

    num_iter = 0
    while num_iter < self.max_iter and pos.count(pos[0]) != len(pos):
        obj = list(map(self.compute_objective_function_value, pos))

        #compute velocity
        v = list(map(self.compute_velocity, pos, v, local_best_pos))

        #update position
        next_pos = list(map(lambda x, y: x + y, pos, v))
        next_obj = list(map(self.compute_objective_function_value, next_pos))

        #update local and global optimum
        local_best_pos, local_best_obj = self.update_local_best(next_pos, next_obj, pos, obj)
        self.update_global_best(local_best_pos, local_best_obj)

        #update position, check whether position are bounded
        pos = list(map(self.bound_value, next_pos))
        num_iter += 1

        print(f"Iteration {num_iter}, global best solution is {round(self.global_best_pos, 4)}, objective value = {round(self.global_best_obj, 4)}")
```

**(c)**

**Result:**

The optimal objective value is 43 when using PSO algorithm.

```
max_iter = 20
num_particles = 4
x_upper = 4
x_lower = -4
```

✓ 0.4s

```
PSO = ParticleSwarmOptimization(max_iter, num_particles, x_upper, x_lower)
PSO.solve()
```

✓ 0.1s

```
Iteration 1, global best solution is 2.0992, objective value = 42.4727
Iteration 2, global best solution is 2.0802, objective value = 42.6602
Iteration 3, global best solution is 2.0802, objective value = 42.6602
Iteration 4, global best solution is 2.0802, objective value = 42.6602
Iteration 5, global best solution is 2.0802, objective value = 42.6602
Iteration 6, global best solution is 2.0802, objective value = 42.6602
Iteration 7, global best solution is 2.0802, objective value = 42.6602
Iteration 8, global best solution is 2.0, objective value = 43.0
Iteration 9, global best solution is 2.0, objective value = 43.0
Iteration 10, global best solution is 2.0, objective value = 43.0
Iteration 11, global best solution is 2.0, objective value = 43.0
Iteration 12, global best solution is 2.0, objective value = 43.0
Iteration 13, global best solution is 2.0, objective value = 43.0
Iteration 14, global best solution is 2.0, objective value = 43.0
Iteration 15, global best solution is 2.0, objective value = 43.0
Iteration 16, global best solution is 2.0, objective value = 43.0
Iteration 17, global best solution is 2.0, objective value = 43.0
Iteration 18, global best solution is 2.0, objective value = 43.0
Iteration 19, global best solution is 2.0, objective value = 43.0
Iteration 20, global best solution is 2.0, objective value = 43.0
```

## Problem2

### (a)

I change the order of **item type** in the weapon list into [Knife, Pistol, Equipment, Primary] for convenience. The order in each item does not changed.

The statement of optimization can be written as

$$max_{x_i \in \{0,1\}} \sum_{i=1}^{n} p_i x_i$$

$$s.t. \sum_{i=1}^{n} w_i x_i \leq 529, \sum_{i=1}^{3} x_i \geq 1, \sum_{i=3}^{6} x_i \geq 1, \sum_{i=6}^{9} x_i \geq 1$$

### (b)

Ans: 6455 (by exhaustive search)

Reference Code:

```python
def exhaustive_search(self):
    candidate_list = list(itertools.product([0, 1], repeat=self.bag_length))
    best_sp = 0
    best_candidate = None
    possible_solutions = 0
    for candidate in candidate_list:
        if self.pass_verification(candidate):
            possible_solutions+=1
            candidate_sp = self.compute_sp(candidate)
            #print(candidate, candidate_sp)
            if candidate_sp > best_sp:
                best_sp = candidate_sp
                best_candidate = candidate

    best_candidate = np.array(best_candidate).astype(bool)
    best_items = np.array(self.items)[best_candidate]
    print(f"Number of all possible solutions = {possible_solutions}")
    print(f"Best survival points is {best_sp}")
    #print(best_candidate)
    print(f"The best items are {best_items}")
```

```python
KSP = KnapSackProblem(max_weight, weight, sp, items)
KSP.exhaustive_search()
```

✓ 0.4s

```
Number of all possible solutions = 6455
Best survival points is 842
The best items are ['Shadow Daggers' 'Desert Eagle Magnum' 'Gas Mask' 'Night-Vision Goggle'
 'Tactical Shield' 'Ingram MAC-10 SMG' 'Leone YG1265 Auto Shotgun'
 'AK-47 Rifle']
```

## (c), (d)

Reference code:

```python
def solve_with_GA(self, pop_size, max_iter):
    num_iter = 0

    global_best_sp = 0
    global_best_candidate = None
    #roulette-wheel selection
    #uniform crossover, prob = 0.1
    #multi bit flip mutation, consecutive based on item types
    population = self.initialize_pop(pop_size)
    population_sp = list(map(self.compute_sp, population))
    best_sp = max(population_sp)
    global_best_sp = best_sp
    global_best_candidate = population[population_sp.index(global_best_sp)]

    while num_iter < max_iter:
        print(f"Iteration {num_iter + 1}, current generation best survival point = {best_sp}, global best survival points is {global_best_sp}")
        parents_a, parents_b = self.roulette_wheel_selection(population)
        offsprings = self.cross_over(parents_a, parents_b)
        population = self.mutation(offsprings)

        #update best candidate
        population_sp = list(map(self.compute_sp, population))
        best_sp = max(population_sp)
        if best_sp > global_best_sp:
            global_best_candidate = population[population_sp.index(best_sp)]
            global_best_sp = best_sp
            best_items = np.array(self.items)[global_best_candidate]

        self.global_best_sp_list.append(global_best_sp)
        num_iter +=1
```

```python
np.random.seed(11)
pop_size = 10
max_iter = 20
KSP = KnapSackProblem(max_weight, weight, sp, items)
KSP.solve_with_GA(pop_size, max_iter)
best_sp_GA = KSP.global_best_sp_list
```

✓ 0.1s Python

```
Iteration 1, current generation best survival point = 296, global best survival points is 296
Iteration 2, current generation best survival point = 505, global best survival points is 505
Iteration 3, current generation best survival point = 294, global best survival points is 505
Iteration 4, current generation best survival point = 526, global best survival points is 526
Iteration 5, current generation best survival point = 598, global best survival points is 598
Iteration 6, current generation best survival point = 474, global best survival points is 598
Iteration 7, current generation best survival point = 666, global best survival points is 666
Iteration 8, current generation best survival point = 800, global best survival points is 800
Iteration 9, current generation best survival point = 622, global best survival points is 800
Iteration 10, current generation best survival point = 681, global best survival points is 800
Iteration 11, current generation best survival point = 542, global best survival points is 800
Iteration 12, current generation best survival point = 722, global best survival points is 800
Iteration 13, current generation best survival point = 684, global best survival points is 800
Iteration 14, current generation best survival point = 384, global best survival points is 800
Iteration 15, current generation best survival point = 674, global best survival points is 800
Iteration 16, current generation best survival point = 803, global best survival points is 803
Iteration 17, current generation best survival point = 515, global best survival points is 803
Iteration 18, current generation best survival point = 289, global best survival points is 803
Iteration 19, current generation best survival point = 712, global best survival points is 803
Iteration 20, current generation best survival point = 286, global best survival points is 803
```

## (e)

Reference code - hill climbing:

```python
def solve_with_hill_climbing(self, max_iter):
    cur_point = self.initialize()
    num_iter = 0
    global_best_sp = self.compute_sp(cur_point)
    global_best_candidate = None

    while num_iter < max_iter:
        print(f"Iteration {num_iter}: best surivival point = {global_best_sp}")
        best_neighbor, best_neighbor_sp = self.find_best_neighbors(cur_point)
        if best_neighbor_sp > global_best_sp:
            global_best_sp = self.compute_sp(best_neighbor)
            global_best_candidate = best_neighbor
            cur_point = best_neighbor
        self.global_best_sp_list.append(global_best_sp)
        num_iter+=1


    best_items = np.array(self.items)[global_best_candidate]
    print(f"Best solution is {best_items}")
```

```
np.random.seed(157)
max_iter = 200
KSP = KnapSackProblem(max_weight, weight, sp, items)
KSP.solve_with_hill_climbing(max_iter)
best_sp_HC = KSP.global_best_sp_list
```

✓ 0.3s Python

```
Iteration 0: best surivival point = 463
Iteration 1: best surivival point = 676
Iteration 2: best surivival point = 705
Iteration 3: best surivival point = 714
Iteration 4: best surivival point = 714
Iteration 5: best surivival point = 714
Iteration 6: best surivival point = 714
Iteration 7: best surivival point = 714
Iteration 8: best surivival point = 714
Iteration 9: best surivival point = 714
Iteration 10: best surivival point = 714
Iteration 11: best surivival point = 714
Iteration 12: best surivival point = 714
Iteration 13: best surivival point = 714
Iteration 14: best surivival point = 714
Iteration 15: best surivival point = 714
Iteration 16: best surivival point = 714
Iteration 17: best surivival point = 714
Iteration 18: best surivival point = 714
Iteration 19: best surivival point = 714
Iteration 20: best surivival point = 714
```

Reference code - random walk:

```
def solve_with_random_walk(self, max_iter):
    cur_point = self.initialize()
    num_iter = 0
    global_best_sp = self.compute_sp(cur_point)
    global_best_candidate = None

    while num_iter < max_iter:
        print(f"Iteration {num_iter}: best surivival point = {global_best_sp}")
        best_neighbor, best_neighbor_sp = self.find_best_neighbors(cur_point)
        if best_neighbor_sp > global_best_sp:
            global_best_sp = self.compute_sp(best_neighbor)
            global_best_candidate = best_neighbor
        cur_point = best_neighbor
        self.global_best_sp_list.append(global_best_sp)
        num_iter+=1

    best_items = np.array(self.items)[global_best_candidate]
    print(f"Best solution is {best_items}")
```

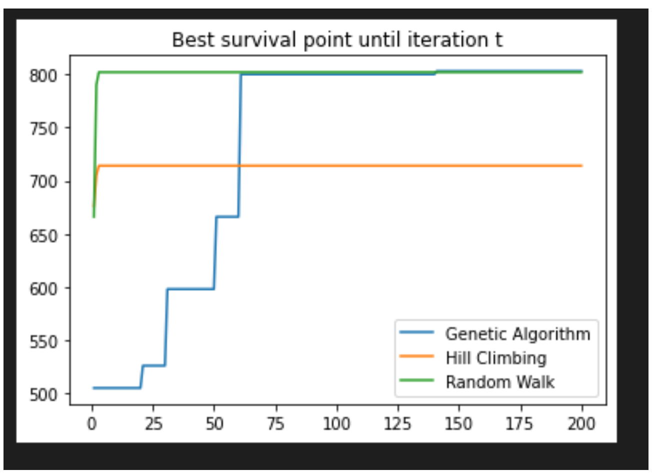
```python
np.random.seed(158)
max_iter = 200
KSP = KnapSackProblem(max_weight, weight, sp, items)
KSP.solve_with_random_walk(max_iter)
best_sp_RW = KSP.global_best_sp_list
```

✓ 0.3s Python

```
Iteration 0: best surivival point = 286
Iteration 1: best surivival point = 666
Iteration 2: best surivival point = 790
Iteration 3: best surivival point = 802
Iteration 4: best surivival point = 802
Iteration 5: best surivival point = 802
Iteration 6: best surivival point = 802
Iteration 7: best surivival point = 802
Iteration 8: best surivival point = 802
Iteration 9: best surivival point = 802
Iteration 10: best surivival point = 802
Iteration 11: best surivival point = 802
Iteration 12: best surivival point = 802
Iteration 13: best surivival point = 802
Iteration 14: best surivival point = 802
Iteration 15: best surivival point = 802
Iteration 16: best surivival point = 802
```

### (f) FE plot

**Observation:**
From the FE plot, we can clearly see that hill climbing tend to stuck in the local optimum. The GA algorithm and random walk algorithm both converge to the best optimum. However, the GA algorithm takes longer time find it.

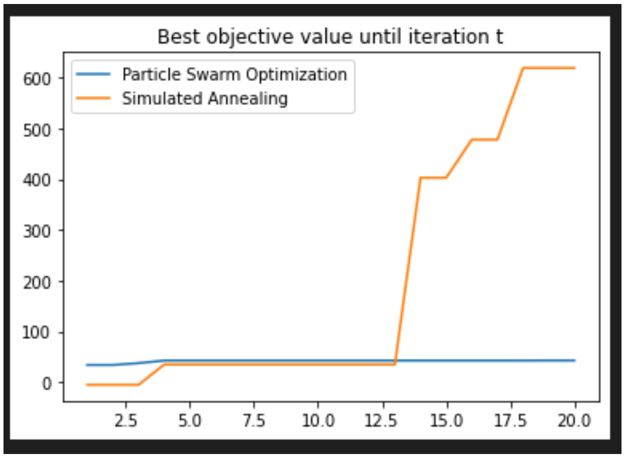



## Problem 3

(I didn't learn linear programming before, so I use the SA algorithm to solve this problem directly.)

## Problem 3-1

**Observation:**
We can clearly see the PSO algorithm tends to stuck in the local optimum (around 2) and the SA algorithm can escape from it due to the temperature mechanism. In conlusion, the SA algorithm seems to be a better choice in this problem.

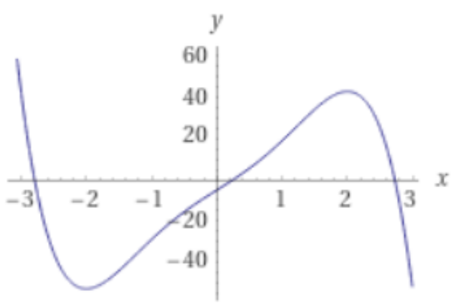



**2 is the local optimum in this objective function**

Input

$$-x^5 + 5x^3 + 20x - 5$$

Plots




(x from −3.1 to 3)

Reference code:

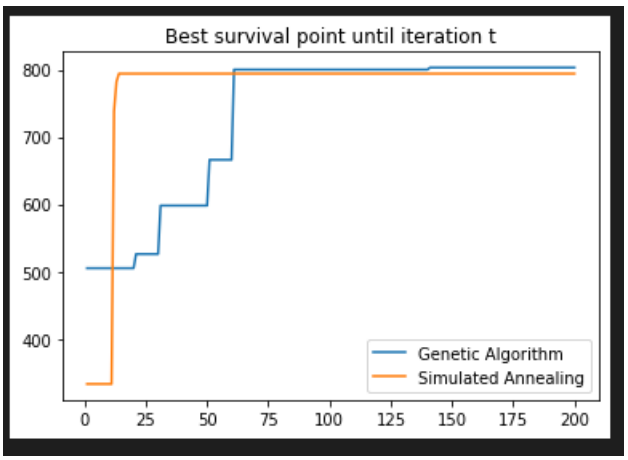
```
class SimulatedAnnealing:
    def __init__(self):
        self.best_obj_val = list()

    def solve(self, init_x, init_temp, mutation_range, max_iter):
        num_iter = 1
        temp_stop = 1 #Temperature lower bound
        #initialization
        cur_temp = self.get_temp(num_iter, init_temp)
        cur_x = init_x
        best_x = cur_x
        cur_val = self.compute_objective_value(cur_x)
        best_val = cur_val
        #start solving
        while num_iter <= max_iter and cur_temp > temp_stop:
            print(f"Iteration {num_iter}, global best solution is {round(best_x, 4)}, objective value = {round(best_val, 4)}")
            mutation_x, mutation_val = self.get_mutation_objective_value(cur_x, mutation_range)
            #print(mutation_x, mutation_val)
            diff = mutation_val - cur_val
            update_prob = self.get_boltzmann_prob(diff, cur_val)
            update_threshold = self.get_uniform_dist()
            if update_prob > update_threshold:
                #record best value
                if mutation_val > best_val:
                    best_val = mutation_val
                    best_x = mutation_x

                #update
                cur_x = mutation_x
                cur_val = mutation_val

            self.best_obj_val.append(best_val)
            num_iter+=1
```

```python
def get_temp(self, num_iter, initial_temp):
    eplison = 0.05
    temp = ((1 - eplison) ** num_iter) * initial_temp
    return temp

def get_boltzmann_prob(self, delta_f, temperature):
    prob = min(1, np.exp(- delta_f / temperature))
    return prob

def get_uniform_dist(self):
    """Get probability from a uniform distribution (0, 1)
    """
    return np.random.rand()

def get_mutation_objective_value(self, x, mutation_range):
    u = self.get_uniform_dist()
    r = (x - mutation_range) + u * (mutation_range * 2)
    #bound value
    r = min(4, r)
    r = max(-4, r)
    mutation_obj_val = self.compute_objective_value(r)
    return r, mutation_obj_val

def compute_objective_value(self, x):
    value = - x ** 5 + 5 * (x ** 3) + 20 * x - 5
    return value
```

## Problem 3-2

**Observation**

The SA algorithm can achieve the same optimum point in the GA algorithm but with a faster path.




Reference code:

```python
def solve_with_simulated_annealing(self, init_temp, max_iter):
    num_iter = 1
    temp_stop = 1 #Temperature lower bound
    #initialization
    cur_temp = self.get_temp(num_iter, init_temp)
    cur_point = self.initialize()
    cur_sp = self.compute_sp(cur_point)
    global_best_candidate = cur_point
    global_best_sp = self.compute_sp(cur_point)
    #start solving
    while num_iter <= max_iter and cur_temp > temp_stop:
        print(f"Iteration {num_iter}: best surivival point = {global_best_sp}")
        best_neighbor, best_neighbor_sp = self.find_best_neighbors(cur_point)
        diff = best_neighbor_sp - cur_sp
        update_prob = self.get_boltzmann_prob(diff, cur_sp)
        update_threshold = self.get_uniform_dist()
        if update_prob > update_threshold:
            #record best value
            if best_neighbor_sp > global_best_sp:
                global_best_sp = best_neighbor_sp
                global_best_candidate = best_neighbor

            #update
            cur_point = best_neighbor
            cur_sp = best_neighbor_sp
        self.global_best_sp_list.append(global_best_sp)
        num_iter+=1

    best_items = np.array(self.items)[global_best_candidate]
    print(f"Best solution is {best_items}")
```