

2019-10-02

# computational Statistics

*HW#3*

192STG11 우나영

# computational Statistics

## HW#3

### 1. Problem

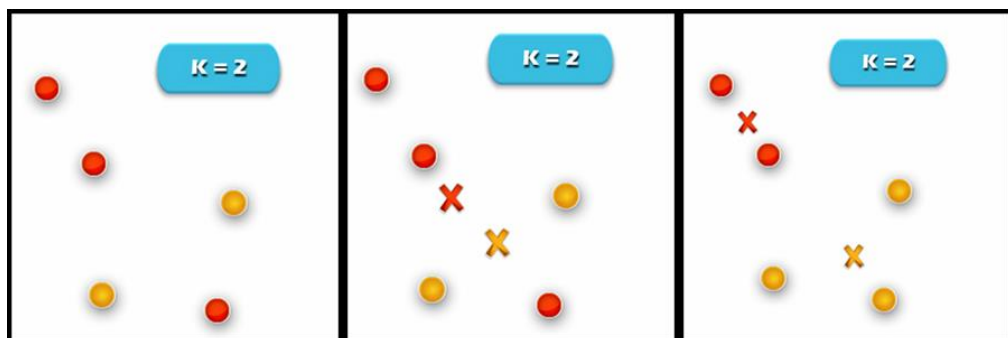
클러스터링 기법이란 유사한 특성을 가진 관측치들을 하나의 그룹으로 묶어주는 기술이다. 클러스터링의 경우 Classification 과 달리 label 없이 관측치들을 분류하기 때문에 machine learning 분야에서는 unsupervised learning 으로 부르기도 한다. 따라서 클러스터링은 EDA 단계에서 데이터의 패턴을 찾기 위해 자주 사용된다. 클러스터링의 주요 이슈는 크게 두가지로 첫번째는 데이터의 클러스터 개수(K)를 정하는 문제와 두번째는 전체 관측치를 K 개의 클러스터로 묶는 경우의 수 문제이다. 첫번째 문제의 경우 해당 데이터의 domain knowledge 를 바탕으로 결정된다. 두번째 문제의 해결방법에 따라 다양한 클러스터링 기법이 개발되었다. 하나의 데이터 내에서 K 개의 클러스터로 나누는 경우의 수는 무수히 많다. Jain 과 Dubes(1988)에 따르면 100 개의 관측치를 4 개의 cluster 로 나누는 경우의 수가 무려  $6.7 \times 10^{58}$ 개이다. 이때 등장하는 것이 바로 local search 이다.

클러스터링에서 가장 빈번히 사용되는 local search 방법은 Kmeans 이다. Kmeans 는 search neighborhood 에서 가장 optimal 한 방향으로 step 을 옮기기 때문에 수렴 속도가 빠르다. 이때 optimal 한 조건은 objective function 인 Within Sum of Squares 를 최소화하는 것이다. 즉 클러스터의 평균과 클러스터 내 관측치들 간의 거리의 합을 줄임으로써 동일한 클러스터 내 관측치들 사이의 거리를 최소화하는 것이다. Kmeans 의 objective function 은 다음과 같다.

$$W(C) = \sum_{k=1}^K \sum_{i=1}^n \|x_i^{(k)} - c_k\|^2 \text{ where } W(C) \text{ is objective function}$$

where K is the number of clusters, n is the number of cases,  $c_k$  is centroid of cluster k

Kmeans 가 optimal 한 클러스터링으로 수렴하는 알고리즘은 다음과 같다.



- step1. cluster 개수인  $K$  가 주어질 때, 전체 관측치 중  $K$  개의 랜덤 포인트를 cluster mean 으로 지정한다.
- step2. 각각의 관측치와 cluster mean 사이의 Euclidean distance 를 구해 cluster 를 나눈다.
- step3. 각각의 cluster 의 새로운 cluster mean 을 구한다. (cluster mean update)
- step4. 다시 각각의 관측치와 cluster mean 사이의 Euclidean distance 를 구해 cluster 를 나눈다. (cluster membership update)
- step5. cluster 의 변동이 없을 때까지 step3 과 step4 를 반복한다.

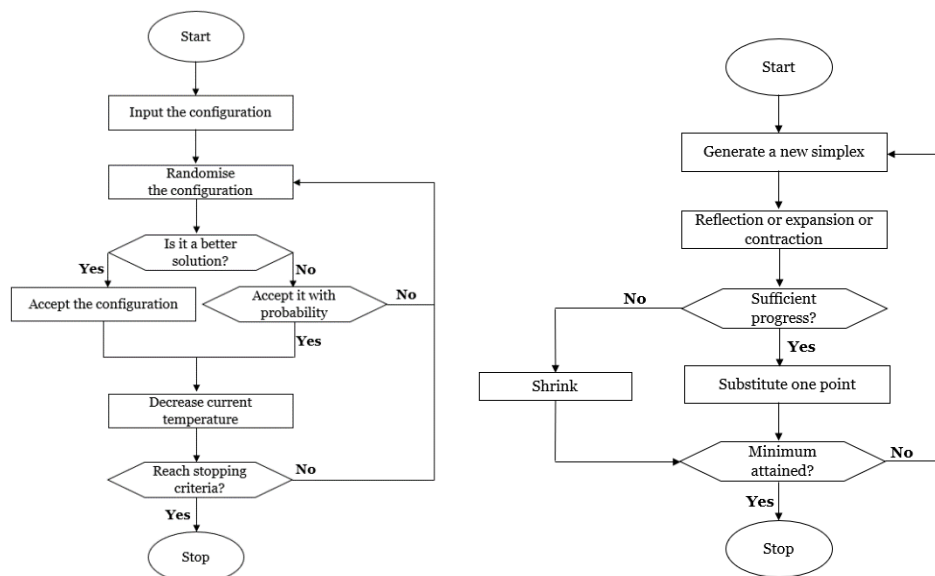
출처: Clustering in R – A Survival Guide on Cluster Analysis in R for Beginners!. BY DATAFLAIR TEAM , JULY 13, 2019

따라서 Kmeans 는 클러스터의 변화가 없을 때까지 1) cluster 의 mean 의 update 와 2) cluster membership 의 update 를 반복적(iterative)으로 수행한다. 위와 같은 알고리즘을 직접 mykmeans 라는 함수로 코딩하여 실행해 볼 것이다.

과제 2 에서 언급했던 Traveling Salesman Problem 과 같은 경우의 수 최적화 문제에서 global search 범위가 천문학적인 단위일 때 우리는 global optima 를 보장할 수는 없으나 한정된 자원으로 경쟁력 있는 local optima 를 구하는 local search 방법을 고안했다. Local search 는 step 의 방향에 따라 steepest 와 random 으로 나뉘는데 steepest 는 현재 시점보다 더 나은 방향으로 step 을 옮기는 방법으로 가장 단시간내에 해에 수렴하기 때문에 heuristic 으로 불리기도 한다. 그러나 heuristic 은 경쟁력 없는 local optima 에 갇혀 더 나은 local optima 로 나아갈 수 없다는 단점이 있다. 이를 보완하기 위해 metaheuristic 은 step 의 방향을 random 으로 설정하여 이미 local optima 에 도달했음에도 불구하고 optimal 하지 않은 방향으로 step 을 옮겨 더 나은 local optima 로 향한다. 따라서 heuristic 과 metaheuristic 은 공통적으로 global optima 를 보장하지는 않지만 제한된 시간 안에 경쟁력 있는 local optima 를 구하는 local search 방법이지만 일단 local optima 에 도달하면 다른 곳으로 이동하지 못하는 heuristic 과 달리 metaheuristic 은 현재의 local optima 을 벗어나 local optima 로 향한다는 차이점이 있다. 이러한 metaheuristic 알고리즘 중 대표적인 방법으로 Simulated Algorithm(SA)과 Genetic Algorithm(GA)이 있다.

Simulated Annealing(SA, 본 과제에서는 optim 의 명칭인 SANN 대신 SA 를 사용할 것이다.)는 금속 공학의 풀림(annealing)에서 아이디어를 얻은 것이다. 풀림이란 금속재료를 적당한 온도로 가열한 다음 서서히 냉각시키는 조작이다. 금속이 가열됨에 따라 원자들이 초기 위치보다 에너지가 높아져 자유로운 상태를 가지지만 이내 냉각되면서 원자들은 원래의 초기 위치보다 더욱 에너지가 낮아진 상태로 더욱 단단하게 굳는다. 이를 SA 에 비유한다면  $\theta \in \Theta$ 는 금속 원자들의 상태이고  $f(\theta)$ 는 금속의

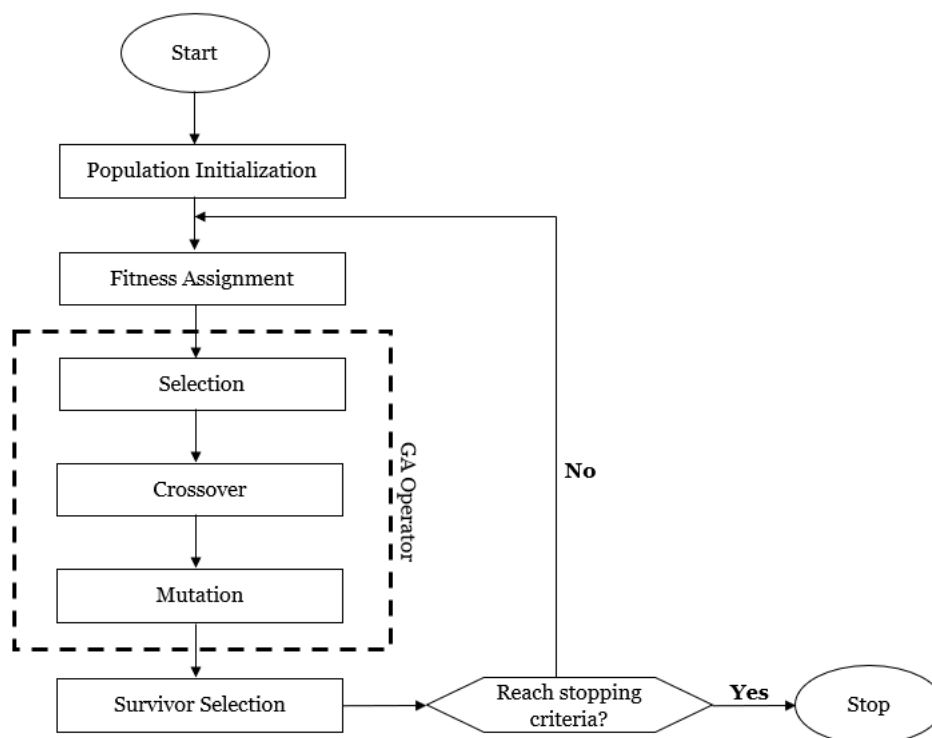
에너지 수준이다. 이때 가장 optimal 한 해는  $f(\theta)$  값을 가장 최소화하는 값이다. SA 는 temperature 에 따라 step 의 방향이 달라진다. Temperature 가 높을 때 마치 금속의 원자들처럼 자유롭게 탐색하면서 objective function 이 optimal 하지 않은 값으로도 이동한다. 따라서 SA 는 local optima 에 도달하더라도 더 나은 local optima 를 위해 현재의 local optima 에서 벗어날 수 있다. 하지만 이내 temperature 가 떨어지면서 금속의 원자들의 운동이 둔해지는 것처럼 search 범위가 좁아져 현재 시점의  $\theta$ 와 가까운 local optima 에 수렴한다. 따라서 SA 가 충분한 search 를 할 수 있도록 temperature 에 따른 cooling schedule 이 잘 설정될 때 global optima 에 수렴할 가능성이 높다. SA 는 R 의 optim 함수를 이용하여 구현할 수 있다. Optim 의 default 방법은 Nelder-Mead 로 simplex 를 이용한 heuristic 방법론 중 하나이다. Result 에서 Nelder-Mead 와 SA 의 최적화 결과를 비교함으로써 heuristic 과 metaheuristic 의 수렴 양상에 어떤 차이가 있는지 알아보자. 또한 SA 의 최적화 결과를 향상하기 위해 temperature 값을 어떻게 설정해야 하는지에 대해서도 알아보자.



좌: SA algorithm 우: Nelder-Mead algorithm

Genetic Algorithm(GA)는 이름처럼 다윈의 자연 선택을 표방한 알고리즘이다. 자연 선택은 개체(individual)들의 형질은 변이(mutation)와 유전 조합(crossover) 과정 속에서 변화하는데 생존에 적합한(fitness) 형질을 가진 개체들이 선택(selection)되어 개체군(population)이 진화한다는 이론이다. GA 에서 개체(individual)는 후보 해(candidate solution)  $\theta$ , 생존 적합성(fitness)는 objective function 인  $f(\theta)$ 의 최적화에 대응된다. 즉 GA 는 후보 해(individual)의 집단(population)이 objective function 을 최적화(fitness)시키는 해의 집단으로 진화하는 과정을 통해 최적화 문제를 해결한다. GA 는 초기 값으로 후보 해(individual)들의 집단(population)을 랜덤하게 선택한다. 이때 iteration 마다 생기는 후보

해(individual)들의 집단(population)을 세대(generation)라 부른다. 세대(generation)마다 개체(individual)들의 적합성(fitness)을 측정한다. 즉 후보 해들의 objective function 값을 계산한다. 이후 자연 선택처럼 후보 해(individual)들을 선택해(selection) 유전자 조합(crossover)과 변이(mutation)를 통해 새로운 세대(generation)를 형성한다. 새로운 세대를 형성하기 위한 일련의 과정(selection-crossover-mutation)을 GA operator 라 부른다. GA 는 이러한 과정을 반복적으로 수행한 후 objective function 을 최적화(fitness)하는 해를 찾거나 maximum iteration 에 도달하면 멈춘다.



## 2. Result

### # Kmeans

아래의 표들은 mykmeans 함수를 실행해 본 결과들이다. 데이터는 R 의 내장된 iris, mtcars, 그리고 Kaggle 의 Filipino Family Income and Expenditure 로 각각 숫자형 설명변수들이 4 차, 10 차 그리고 23 차이다. mtcars 는 반응변수인 mpg 를 5 개의 범주로 나눠 label 을 지정했다. (범주의 기준은 아래 표를 참고) Filipino Family Income and Expenditure 데이터의 경우 label 로 사용된 Region 에 따라 50 개씩 랜덤하게 샘플링하여 데이터의 총 관측치 수를 줄였다. 각각의 결과를 통해 R 에 내장된 kmeans 함수와 mykmeans 함수의 성능 차이를 비교해볼 수 있다.

## ex1 ) iris data

X1	Sepal.Length
X2	Sepal.Width
X3	Petal.Length
X4	Petal.Width
Y	Species
n	150
p	5

iris( 4 dimension )		Kmeans			mykmeans		
		group			group		
		1	2	3	1	2	3
group size		38	50	62	50	38	62
cetroids	X1	6.85	5.006	5.902	5.006	6.85	5.902
	X2	3.074	3.428	2.748	3.428	3.074	2.748
	X3	5.742	1.462	4.394	1.462	5.742	4.394
	X4	2.071	0.246	1.434	0.246	2.071	1.434
withness		23.879	15.151	39.821	15.151	23.879	39.821
tot.withness		78.851			78.851		
heterogeneity(BSS/TSS)		88.4%			88.4%		
iteration		2			2		
nstart		125			125		
time		0.02			1.39		

iris 데이터의 경우 kmeans 는 nstart 가 1 일때도 위의 표와 같은 local optima 에 수렴하는 반면 mykmeans 는 불안정하여 nstart 가 1 일때는 수렴 값이 계속 바뀐다. nstart 를 125 정도로 설정했을 때 비로소 kmeans 와 같은 결과가 나온다. 하지만 mykmeans 의 불안정성으로 인해 nstart 가 125 일 때도 랜덤 초기값에 영향을 받아 optima 결과가 달라진다. 게다가 nstart 를 늘려주게 되면 알고리즘이 작동하지 않는다. 그럼에도 불구하고 mykmeans 는 nstart 가 125 일 때 kmeans 와 같은 결과를 가지는 경우가 있다. 런 타임의 경우에 kmeans 가 mykmeans 에 비해 70 배 빠르다.

## ex2) mtcars

X1	cyl	X7	vs
X2	disp	X8	am
X3	hp	X9	gear
X4	drat	X10	carb
X5	wt	n	32
X6	qsec	p	10

mpg	15 미만	15 이상 20 미만	20 이상 25 미만	25 이상 30 미만	30 이상
group	1	2	3	4	5
group size	5	13	8	2	4

mtcars( 10 dimension )		Kmeans					mykmeans				
		group					group				
group size		1	2	3	4	5	1	2	3	4	5
		4	7	10	4	7	4	5	7	10	6
centroids	X1	8	7.714	4	8	5.714	8	8	5.714	4	7.667
	X2	340.5	295.343	101.57	443	166.57	340.5	426.4	166.57	101.57	284.567
	X3	272.25	160.714	81.4	206.2	120.14	272.25	200	120.14	81.4	158.333
	X4	3.675	3.05	4.086	3.06	3.708	3.675	3.078	3.706	4.086	3.033
	X5	3.538	3.599	2.199	4.966	3.108	3.538	4.661	3.108	2.199	3.625
	X6	15.088	17.661	18.761	17.568	18.471	15.088	17.46	18.471	18.761	17.768
	X7	0	0.143	0.9	0	0.571	0	0	0.571	0.9	0.167
	X8	0.5	0	0.8	0	0.429	0.5	0	0.429	0.8	0
	X9	4	3	4.1	3	4	4	3	4	4.1	3
	X10	5	2.286	1.5	3.5	3.571	5	3.2	3.571	1.5	2.333
withinness		7650.7	11443.34	10060	4612	8789	7650.7	10908	8789	10060	6327.2
tot.withiness		42555.66					43735.89				
heterogeneity(BSS/TSS)		93.2%					93.0%				
iteration		2					2				
nstart		10					10				
time		0.0019					0.08				

mtcars 데이터의 경우 mykmeans 는 nstart 가 10 까지만 수렴한다. 그 이후로는 작동하지 않는다. nstart 가 10 일 경우 kmeans 와 비슷한 결과 값을 보여준다. 그러나 mykmeans 의 불안정성으로 인해 nstart 가 10 일 때 수렴의 여부와 수렴 값이 달라진다.

### ex3) Filipino Family Income and Expenditure

X1	income	X9	Restaurant.and.hotels. Expenditure	X17	Communication. Expenditure
X2	Total.Food. Expenditure	X10	Alcoholic.Beverages. Expenditure	X18	Education. Expenditure
X3	Bread.and.Cereals. Expenditure	X11	Tobacco. Expenditure	X19	Miscellaneous.Goods.and. Services.Expenditure
X4	Total.Rice. Expenditure	X12	Clothing..Footwear.and. Other.Wear.Expenditure	X20	Special.Occasions. Expenditure
X5	Meat. Expenditure	X13	Housing.and.water. Expenditure	X21	Crop.Farming.and. Gardening.expenses
X6	Total.Fish.and.. marine.products.Expenditure	X14	Imputed.House. Rental.Value	X22	Total.Income.from. Entrepreneurial.Acitivites
X7	Fruit. Expenditure	X15	Medical.Care. Expenditure	X23	Household.Head.Age
X8	Vegetables. Expenditure	X16	Transportation. Expenditure	Y	Region
n	850	p	23		

filipino( 23 dimension )	Kmeans	mykmeans
tot.withiness	6.506728e+12	1.197973e+13
heterogeneity(BSS/TSS)	91.1%	81.6%
iteration	10	2
nstart	15	15
time	0.13	5.94

group	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
kmeans	1	93	78	1	4	54	8	25	2	39	48	57	111	132	10	154	33
mykmeans	44	20	20	72	3	21	84	39	6	25	123	63	78	51	2	72	127

Filipino Family Income and Expenditure 데이터의 경우 cluster 가 17 개이기 때문에 가독성을 위해 centroid 와 개별 cluster 의 withiness 를 첨부하지 않았다. Kmeans 의 경우 iter.max 의 디폴트 값인 10 번 이내에 수렴하지 않았다. 설명변수의 차원이 높아짐에 따라 점점 Kmeans 와 mykmeans 의 결과값이 달라짐을 확인할 수 있다. 특히 cluster 간의 이질성의 척도인 heterogeneity 을 근거로 했을 때 mykmeans 가 Kmeans 에 비해 성능이 떨어짐을 확인할 수 있다. 런 타임에서도 kmeans 가 mykmeans 보다 우위에 있다. 이전의 예제와 비교해봤을 때 이번 예제에서 관측치 수와 cluster 수가 늘어남에 따라 mykmeans 는 알고리즘이 불안정하기 때문에 훨씬 런 타임이 오래 걸린 다는 것을 알 수 있다.

#### # Simulated Annealing(SA)

R 의 optim 함수는 multiple dimension 의 최적화에 사용되는 함수로 최적화 방법은 Nelder-Mead, BFGS, CG, L-BFGS-B, SANN, Brent 가 있으며 default 는 Nelder-Mead 이다. SA 는 optim 의 control argument 를 통해 parameter 들을 지정할 수 있다.

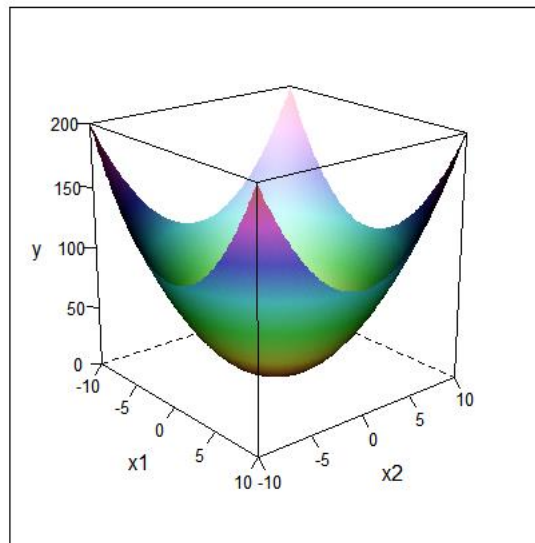
parameter	설명
maxit	default 는 10000 이다. 최대 iteration 횟수이다. SA 는 stopping criterion 이 없기 때문에 maxit 이 stopping rule 이 된다.
temp	default 는 10 이다. cooling schedule 의 초기 temperature 값이다.
tmax	default 는 10 이다. 각각의 temperature stage 에서 발생하는 iteration 횟수이다.

SA 의 temp 와 tmax 으로 cooling schedule 조절함으로써 optimization 성능을 조절할 수 있다. 아래 예제를 통해 SA 와 Nelder-Mead 의 수렴 양상의 차이점을 비교하고 SA 의 성능을 높이기 위해 parameter 값을 어떻게 설정해야 할지 알아보자.



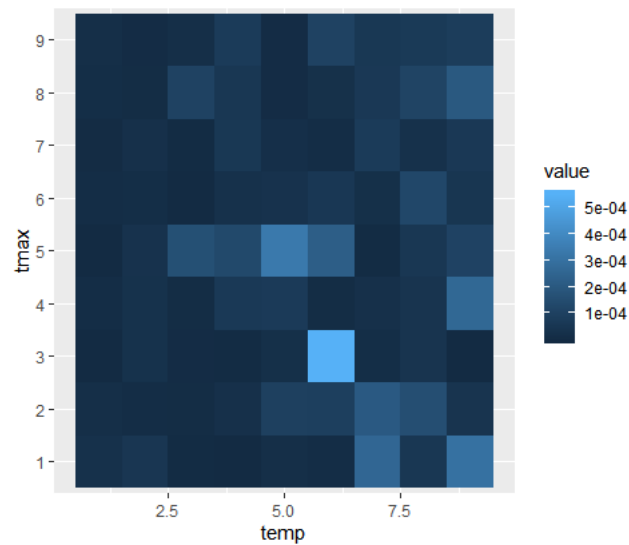
ex1. Sphere function

Sphere function



method	Nelder-Mead	Simulated Annealing	
initial=c(x1,x2)	c(1,1)	c(1,1)	c(1,1)
$\theta = c(x1, x2)$	c(3.75E-05, 5.17E-05)	c(-9.E-03, -1.E-03)	c(0.002, 0.0019)
f( $\theta$ *)	4.09E-09	8.79E-05	8.230047e-06
time	0.003	0.45	0.61
temp		10	3
tmax		10	6

sphere 함수는 global minimum 이 하나 있는 함수이다. sphere 함수에서 SA 와 Nelder-Mead 둘 다 objective function 의 값이 0 근처에 수렴하였다. 그러나 예상과 달리 Nelder-Mead 함수가 SA 보다 더 낮은 objective function 의 값으로 수렴하였다. 또한 Nelder-Mead 가 SA 보다 런 타임의 속도에 있어서 우위에 있었다. 한편 SA 의 temp 와 tmax 의 값을 조정하면 objective function 의 값을 낮출 수 있었다. Objective function 을 최소화시키는 temp 와 tmax 값의 조합을 찾아본 결과 이 함수의 경우 temp 가 3 이고 tmax 가 6 일 때였다. 따라서 temp 를 3 그리고 tmax 를 6 으로 지정했을 때 temp 와 tmax 가 default 값일 때보다 objective function 이 낮아진 것을 확인할 수 있다.



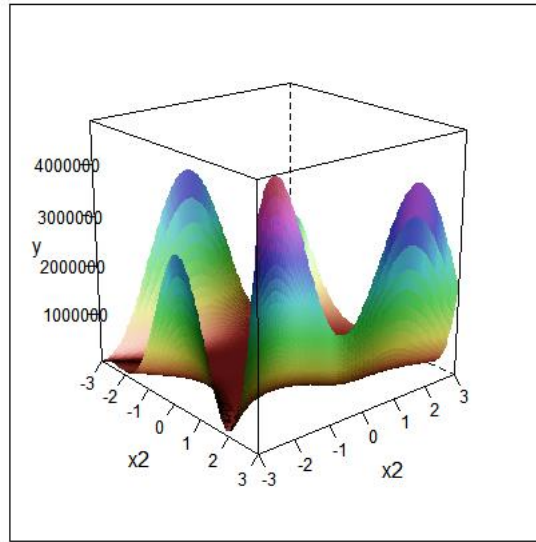
위의 그래프는 temp 와 tmax 의 조합에 따른 objective function 값이다. temp 와 tmax 의 범위는 temp 와 tmax 의 값이 default 값인 10 일 때보다 작을 때 objective function 이 최소화된다는 것을 관찰한 것을 바탕으로 1 에서 9 사이의 양의 정수로 정하였다. 위 그래프에 따르면 주어진 범위 내에서 temp 가 3 이고 tmax 가 6 일 때 objective function 이 가장 최소화됨을 알 수 있다. 하지만 SA 의 cooling schedule 은 위의 방법보다 좀 더 기술적인 방법을 사용하여 parameter 를 tuning 할 필요가 있다. 위의 그래프는 이상적인 cooling schedule 을 찾기 위한 미숙한 시도였음을 밝힌다.

method	Nelder-Mead	Simulated Annealing
$\theta = c(x_1, x_2)$	$c(1.325267e-09, -2.680448e-09)$	$c(-6.e-05, -2.e-04)$
$f(\theta^*)$	8.941133e-18	3.149915e-08
temp		3
tmax		6

Computing capacity 가 허락하는 범위내에서 최적화 실행 후 발생한 해를 다음 최적화 실행의 초기값으로 설정해 연쇄적으로 실행하였다. Nelder-Mead 은 초기값이  $c(1,1)$ 일 때 objective function 이  $4.09E-09$  이었지만 연쇄적인 실행 결과 objective function 이  $8.941133e-18$  까지 감소했다. SA 는 초기값이  $c(1,1)$ 일 때 objective function 이  $8.230047e-06$  이었지만 연쇄적인 실행 결과 objective function 이  $3.149915e-08$  까지 감소했다. 더 감소할 여지가 있겠지만 computing capacity 의 한계로 더 이상 최소화된 값을 구할 수는 없었다. 위의 결과를 바탕으로 Nelder-Mead 는 초기값을 바꿈에 따라 objective function 의 값이 최소화되는 폭이 SA 보다 크기 때문에 초기값에 영향을 많이 받는다는 것을 알 수 있다.

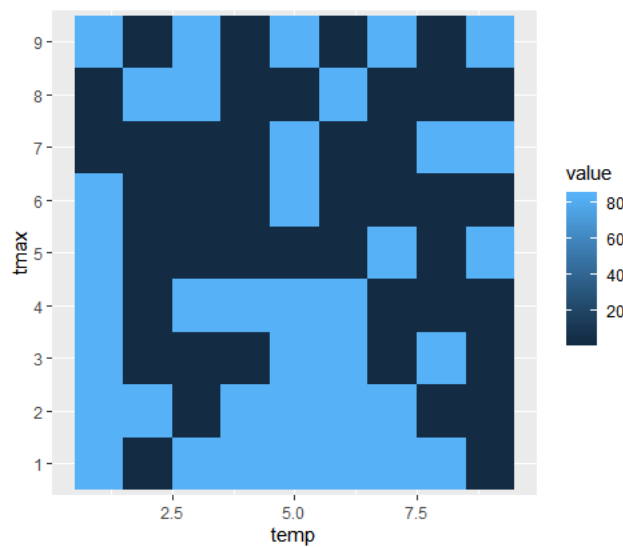
ex2. Goldstein Price function

Goldstein Price function



method	Nelder-Mead		Simulated Annealing				
initial=c(x1,x2)	c(1,1)	c(1.2, 0.8)	c(1,1)	c(1.78, 0.2)	c(1,1)	c(-0.0002, -0.99)	c(-0.0002, -0.99)
$\theta^*=c(x1, x2)$	c(1.2, 0.8)	c(1.2, 0.8)	c(1.78, 0.2)	c(1.8, 0.2)	c(-0.0002, -0.99)	c(-0.0002, -0.99)	c(-0.0002, -0.99)
f( $\theta^*$ )	840	840	84	84	3	3	3
time	0.007	0.005	0.08	0.06	0.08	0.06	0.08
temp			10	10	1	10	1
tmax			10	10	8	10	8

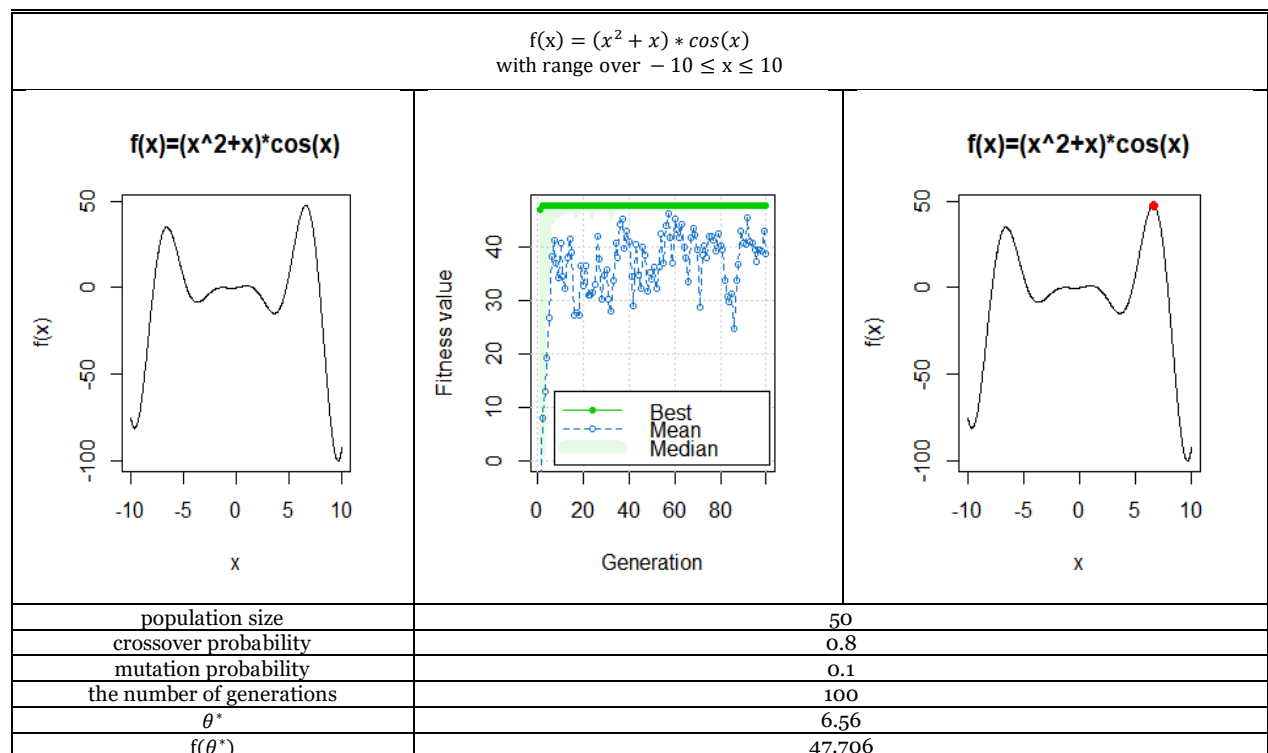
예제 2 번 함수의 경우 예제 1 보다 굴곡이 커 global minima 를 찾는 것이 까다롭다. 이 경우 Nelder-Mead 와 SA 에 동일한 초기값인  $c(1,1)$ 을 설정했을 때 SA 가 Nelder-Mead 보다 objective function 이 최소인 값에 수렴하였다. 예제 1 과 같은 방법으로 Nelder-Mead 에 이전에 수렴한 값으로 초기값을 설정했음에도 불구하고 동일한 objective function 값으로 수렴했다. 따라서 이를 통해 Nelder-Mead 는 굴곡이 심한 함수에서 local optima 에 수렴하는 경우 기존의 local optima 에서 벗어나기 힘들다는 것을 확인할 수 있다. SA 도 예제 1 과 같이 초기값을 이전에 수렴한 값으로 설정했을 때 동일한 objective function 값으로 수렴했다. 하지만 SA 의 temp 와 tmax 를 수정 후 objective function 이 더 낮은 값으로 수렴하였다. 따라서 SA 의 경우 초기값보다 temp 와 tmax 를 이용해 cooling schedule 을 조절하는 것이 objective function 을 최소화하는데 효과적이다. temp 와 tmax 는 예제 1 과 동일한 방법으로 구하였다. 아래는 temp 와 tmax 의 조합에 따른 objective function 값을 그래프로 나타낸 것이다. 그래프에 따르면 temp 가 1 이고 tmax 가 8 일 때 objective function 을 가장 최소화할 수 있다. 또한 SA 의 런 타임은 예제 1 과 같은 스무스한 function 보다 예제 2 와 같은 굴곡진 함수에서 더 빠르다는 것을 알 수 있다. 따라서 이번 예제에서 Nelder-Mead 와 SA 의 결과를 살펴봄으로써 local optima 에 갇히는 heuristic 과 달리 기존의 local optima 에서 벗어나 global optima 로 향하는 metaheuristic 알고리즘의 특성을 비교할 수 있다. 또한 SA 는 초기값보다는 cooling schedule 이 수정된 후 성능이 향상되었다.



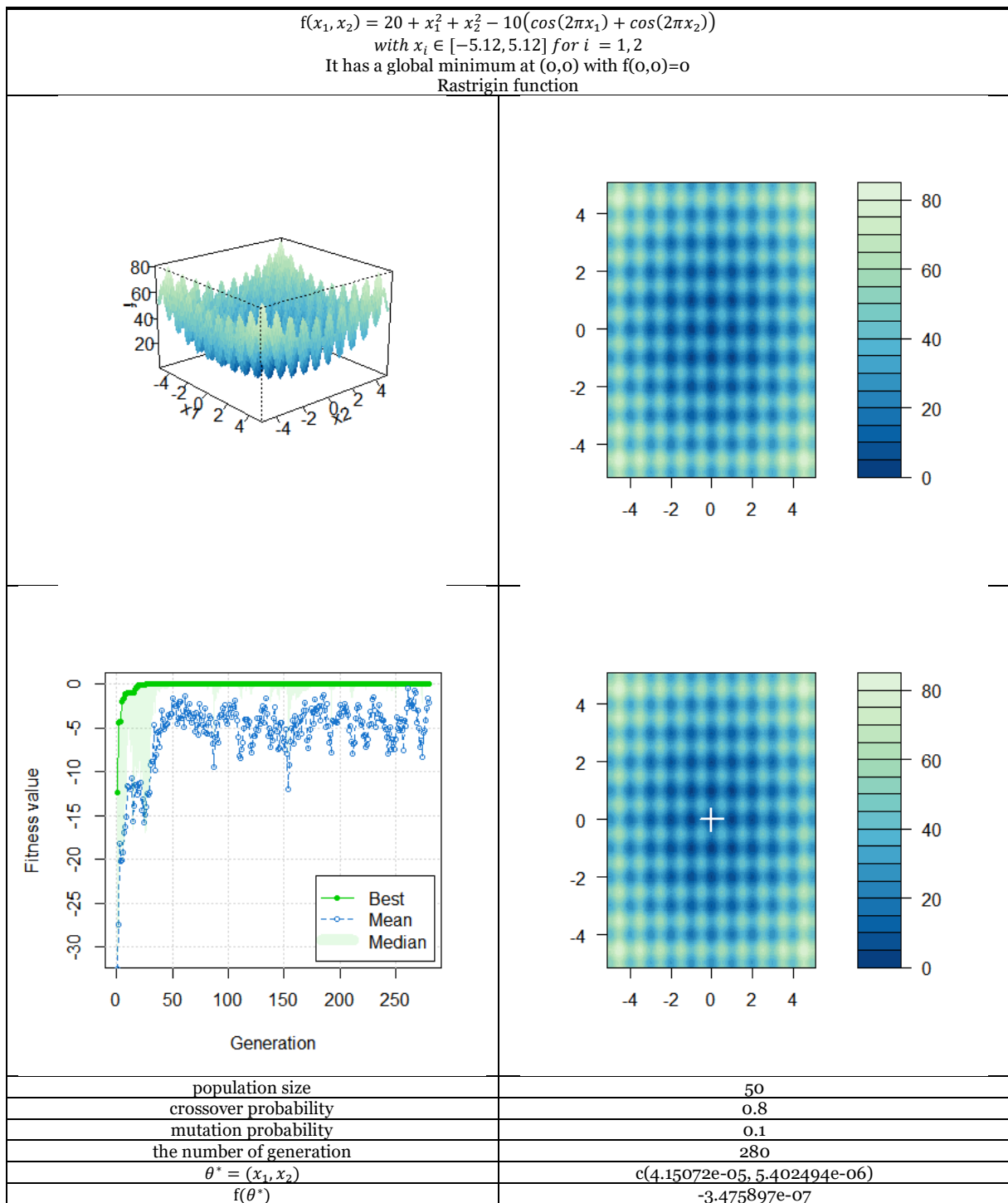
# Genetic Algorithm(GA)

<https://cran.r-project.org/web/packages/GA/vignettes/GA.html> 의 내용을 참고하여 GA 를 이용한 최적화를 실현해 볼 것이다.

Function optimization in one dimension

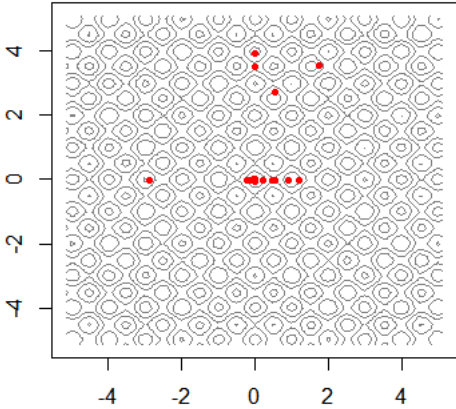


Function optimization in two dimension



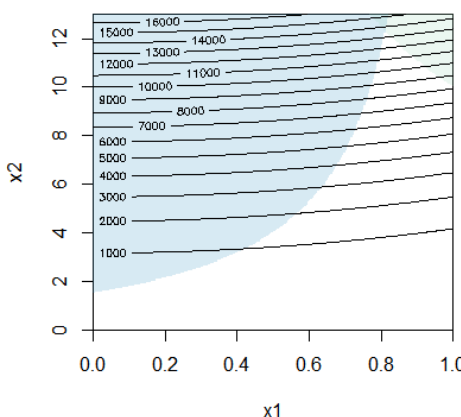
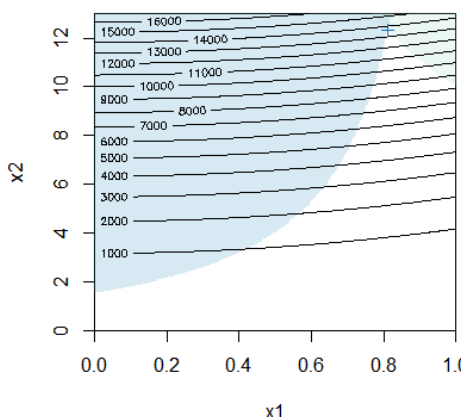
Rastrigin function 은 local minima 가 많아 optimization algorithm 을 테스트하기에 좋은 함수 중 하나이다. global minima 는  $f(0,0)=0$  이다. fitness function value 와 solution 이 global minima 에 근접한 값에 수렴한다는 것을 확인할 수 있다.

ga 함수의 유용한 argument 들

monitor	suggest
<p>iteration= 100</p> 	<pre>&gt; GA1 &lt;- ga(type="real-valued",   fitness = function(x)-Rastrigin(x[1],x[2]),   lower=c(-5.12, -5.12), upper=c(5.12, 5.12),   suggestions=suggestedSol,   popSize=50, maxiter=1)  &gt; head(GA1@population) [.1]      [.2] [1,]  0.200000  1.500000 [2,] -1.500000  0.500000 [3,]  2.225604 -1.9651927 [4,]  3.731019 -2.9575058 [5,] -3.489729  2.1571834 [6,] -3.963978  0.1046975</pre>

ga 의 함수의 유용한 argument 를 소개한다. 첫번째로 monitor argument 는 GA 의 수렴 프로세스를 애니메이션으로 확인할 수 있다. 위의 Rastrigin 예제를 monitor 를 통해 살펴본 결과 global minima 인 중간 지점으로 빨간 점들이 모여든다. suggest argument 를 이용해 후보 해를 사용자가 지정할 수도 있다. 그 외 나머지 후보 해는 원래 알고리즘 대로 random 하게 선택된다.

Constrained Optimization

$\min_x f(x) = 100(x_1^2 - x_2)^2$ <p>subject to following nonlinear inequality constraints and bounds</p> $x_1 x_2 + x_1 - x_2 + 1.5 \leq 0$ $10 - x_1 x_2 \leq 0$ $0 \leq x_1 \leq 1, 0 \leq x_2 \leq 13$	
	
population size	50
crossover probability	0.8
mutation probability	0.1

the number of generation	339
$\theta^* = (x_1, x_2)$	c(0.8120632, 12.31468)
$f(\theta^*)$	-13584.49

이번 예제에서 GA 를 이용하여 비선형 제약 조건과 구간을 만족하는 동시에 objective function 을 최소화하는 해를 찾을 수 있었다. 위의 그래프에서 음영이 들어간 구간이 비선형 제약 조건을 만족하는 후보 해들의 집합이고 global optima 가 “+”로 표시되었다.

GA 의 후보 해(candidate solution)들의 형질(gene)을 코딩할 수 있는 방법은 크게 binary 와 real-valued 로 나뉜다. 다음의 예제는 loss function 을 최소화하는 sample size 와 acceptance number 를 구하는 문제이다. 그러나 최적화 문제의 결과보다 후보 해의 코딩 방법에 따른 GA 의 fitted value 값, 수렴속도 그리고 수렴 횟수(iterations)를 비교해볼 것이다. 아래는 각각의 코딩방법에 따라 GA 를 100 번씩 수행한 결과의 평균값이다. 이를 통해 real-valued 가 binary 만큼 정확한 동시에 수렴 속도와 수렴 횟수의 측면에서 우위에 있음을 알 수 있다.

	binary	real-valued
수렴 속도	1.939	0.862
수렴 횟수	161.04	129.74
fitted value	0.09382	0.096

Hybrid GA 는 GA 방법과 다른 local search 방법을 결합한 알고리즘으로 최적화 성능을 높일 수 있다는 장점이 있다. Rastrigin 예제에서 GA 와 optim 의 L-BFGS-B 을 결합해 적용해본 결과 population size 는 50, crossover probability 는 0.8, mutation probability 는 0.1, the number of generation 은 183,  $\theta^* = c(0,0)$  이고  $f(\theta^*) = 0$  이다. 예제 2 에서 GA 만 사용했을 때보다 hybrid GA 를 사용했을 때 Rastrigin 의 global minima 에 더 정확히 수렴할 수 있다.

### 3. Discussion

mykmeans 는 Kmeans 의 알고리즘에 따라 코딩을 했음에도 불구하고 코딩의 미숙함으로 인해 R 의 내장 함수인 Kmeans 에 비해 수렴하는 것이 불안정했다. 수렴이 불안정한 원인으로 두가지를 지적할 수 있다. 첫번째는 mykmeans 가 Kmeans 에 비해 더 랜덤 initial point 에 영향을 많이 받기 때문에 수렴 값의 변동폭이 컸다. 그러나 nstart 의 값이 커질수록 kmeans 의 결과값과 유사해졌다. 이때 nstart 는 kmeans 알고리즘을 random start 의 횟수로 nstart 개의 클러스터 결과 중에 가장 Within SS 가 작은 클러스터를 선택할 수 있다. 두번째로는 colMeans 함수의 dimension 문제이다. nstart 가 커질수록 random initial value 가 많아진다. initial value 에 따라서는 수렴하는 과정에서 특정 cluster 는 관측치가 없을 수 있다. 이러한 경우 dimension 이 0 이 되어 colMeans 함수는 cluster means 을 update 할 수 없기 때문에 오류가 발생하는 것으로 추측된다. 이에 따라 mykmeans 는 데이터의 dimension 이 높을 경우 initial point 의 경우의 수가 많아져 랜덤 initial point 에 영향을 더 많이 받게 되고 optimal 한 클러스터를

구하기 위해 `nstart` 의 값을 키울수록 `colMeans` 오류가 수반되어 R 의 `Kmeans` 와 비교해 보았을 때 성능이 현저히 떨어지게 된다.

R optim 의 Nelder-Mead 와 SA 를 비교함으로써 local search 방법론 중에서도 heuristic 과 metaheuristic 의 차이점을 공부할 수 있었다. 특히 그 차이점은 smooth 한 function 보다 변곡점과 local optima 가 많은 function 에서 극대화되었다. Nelder-Mead 는 일단 local optima 에 수렴하면 다른 local optima 로 벗어나기 어려운 반면 SA 는 cooling schedule 의 조절을 한다면 기존의 local optima 를 벗어나 더 나은 최적의 해에 수렴할 수 있었다. SA 는 cooling schedule 에 따라 성능이 좌우되기 때문에 cooling schedule 의 parameter 인 temperature 를 결정하는 기술적인 방법이 요구된다.

GA 도 metaheuristic 방법론 중 하나로 다윈의 자연선택을 차용한 알고리즘이다. GA 는 요약하면 후보 해들의 집단을 랜덤하게 선택한 후 각각의 후보 해들에 대한 objective function 의 값을 계산해 후보 해들을 선택한다. 선택된 후보 해들의 유전자 조합과 변이를 통해 새로운 후보 해들의 집단으로 만드는 과정을 반복함으로써 초기의 후보 해 집단은 objective function 을 최적화하는 해의 집단으로 진화해 나간다. GA 만으로도 뛰어난 성능을 보이지만 다른 local search 방법론이랑 결합되었을 때 더 성능을 높일 수 있다.

## 4. Appendix

```
rm(list=ls())
library(tidyverse)
#1. my own Kmeans function
set.seed(0929)

dist<- function(p, q){
  sum((p-q)^2)
}

mykmeans <- function(x, K, iter.max=10, nstart=1){

  history <- list()
  history.wss <- NULL

  for( i in 1:nstart){
    # initial values
    x <- as.matrix(x)
    current <- x[sample(nrow(x),K), ]
    niter <- 0
    err <- 1
    grp <- rep(0, nrow(x))

    while(niter <= iter.max && err!=0){

      # update K cluster membership by using Euclidean distance
      d <- NULL
      for(j in 1:K){
        euc <- apply(x, 1, function(x)dist(x, current[j,]))
```



```

        d <- cbind(d, euc)
    }
    old.grp <- grp
    grp <- apply(d, 1, which.min)

    # update K cluster means
    new <- NULL
    for(m in 1:K){
        a <- colMeans(x[grp==m,])
        new <- rbind(new, a)
    }

    # update niter and error
    niter <- niter + 1
    err <- sum(old.grp-grp)
}

# WSS and TSS and size
wss <- NULL
for(j in 1:K)wss[j] <- sum(apply(x[grp==j,],1, function(x)dist(x,new[j,])))
WSS <- sum(wss)
TSS <- sum(apply(x, 1, function(xx)dist(xx, colMeans(x))))
BSS <- TSS-WSS
size <- table(grp)
rownames(new) <- NULL

history.wss[i] <- WSS
history[[i]] <- list(clusters=grp, centers=new, iter=niter, withiness=wss,
tot.withiness=WSS, betweeness=BSS,
                    totss=TSS, heterogeneity=BSS/TSS, size=size)

}

return(history[[which.min(history.wss)]])
}

# example 1. iris
system.time(kmeans(iris[, -5], 3, nstart=125))
system.time(mykmeans(iris[, -5], 3, nstart=125))

# example 2. 10 dimension
mtcars$grp <- ifelse(mtcars$mpg<15, 1,
                    ifelse(mtcars$mpg<20, 2,
                            ifelse(mtcars$mpg<25, 3,
                                    ifelse(mtcars$mpg<30, 4, 5))))

table(mtcars$grp)
system.time(mykmeans(mtcars[, -c(1,12)], 5, nstart=10))
system.time(kmeans(mtcars[, -c(1,12)], 5, nstart=10))

# 20 dimension
fp <- read.csv('C:/Users/dnskd/Desktop/19-2/계특/과제/hw3/fp.csv')
region <- unique(fp$Region)

smp <- list()
for (i in 1:length(region)){
    df <- fp %>% filter(fp$Region==region[i])
    smp[[i]] <- df[sample(nrow(df), 50), ]
}
temp <- do.call(rbind, smp)
smp <- temp

```

```
table(smp$Region)
system.time(kmeans(smp[, -2], 17, nstart=15))
system.time(mykmeans(smp[, -2], 17, nstart=15))

library(lattice)
options(scipen=999)

# ex 1. sphere function

f.name <- "Sphere function"

# define the function
f.sphere <- function(x){
  x <- matrix(x, ncol=2)
  f.x <- apply(x^2, 1, sum)
  return(f.x)
}

# plot the function
x1 <- seq(-10, 10, length=101)
x2 <- seq(-10, 10, length=101)
X <- as.matrix(expand.grid(x1, x2))
colnames(X) <- c("x1", "x2")
y <- f.sphere(X)
df <- data.frame(X, y)

wireframe(y ~ x1*x2,
           data= df,
           main=f.name,
           shade=T,
           scales=list(arrows=F),
           screen=list(z=-50, x=-70))

# Nelder-Mead
nm <- function(par){

  output <- 10000 # for while loop
  thr <- 10^(-17)

  while(output>thr){
    result <- optim(par, f.sphere, method="Nelder-Mead")
    output <- result$value
    par <- result$par
  }
  return(result)
}

nm(par=c(1,1))

system.time(for(i in 1:100) optim(par=c(1,1), f.sphere, method="Nelder-Mead"))

# Simulated-Annealing
system.time(optim(par=c(1,1), f.sphere, method="SANN"))

obj <- matrix(0, ncol=9, nrow=9)
var1 <- matrix(0, ncol=9, nrow=9)
var2 <- matrix(0, ncol=9, nrow=9)

for(i in 1:9){
  for(j in 1:9){
    output <- optim(par=c(1,1), f.sphere, method="SANN", control=list(temp=i, tmax=j))
```

```

    obj[i,j] <- output$value
    var1[i,j] <- output$par[1]
    var2[i,j] <- output$par[2]
  }
}
temp <- 1:9; tmax <- 1:9
obj <- as.data.frame(obj)
colnames(obj) <- 1:9
obj <- obj %>% gather('1':'9', key="tmax", value="value")
obj$temp <- rep(1:9,9)
obj %>% ggplot(aes(temp, tmax))+geom_tile(aes(fill=value))
which.min(obj$value) # temp=3, tmax=6

system.time(optim(par=c(1,1), f.sphere, method="SANN", control=list(temp=3, tmax=6)))

sa <- function(par){

  output <- 10000 # for while loop
  thr <- 10^(-7)

  while(output > thr){
    result <- optim(par, f.sphere, method="SANN", control=list(temp=3,tmax=6))
    output <- result$value
    par <- result$par
  }
  return(result)
}
sa(par=c(1,1))

# ex2. Goldstein-Price function

f.name <- "Goldstein Price function"

# define the function
f.GP <- function(x){
  x <- matrix(x, ncol=2)
  x1 <- x[,1]
  x2 <- x[,2]
  f.xy <- (1+(x1+x2+1)^(2)*(19-14*x1+3*x1^(2)-14*x2+6*x1*x2+3*x2^(2)))*(30+(2*x1-
3*x2)^(2)*(18-32*x1+12*x1^(2)+48*x2-36*x1*x2+27*x2^(2)))
  return(f.xy)
}
x1 <- seq(-3, 3, length=101)
x2 <- seq(-3, 3, length=101)

X <- as.matrix(expand.grid(x1,x2))
colnames(X) <- c("x1", "x2")
y <- f.GP(X)
df <- data.frame(X,y)

# plot the function
wireframe(y ~ x2 *x2 , data=df, main=f.name,
          shade=T, scales=list(arrows=F), screen=list(z=-50, x=-70))

# Nelder-Mead
res1 <- optim(c(1,1), f.GP, method="Nelder-Mead")
res1
res2 <- optim(par=res1$par, f.GP, method="Nelder-Mead")
res2
system.time(for(i in 1:100)optim(c(1,1), f.GP, method="Nelder-Mead"))
system.time(for(i in 1:100)optim(res1$par, f.GP, method="Nelder-Mead"))

```

```

# Simulated Annealing
set.seed(1234)
obj <- matrix(0, ncol=9, nrow=9)
var1 <- matrix(0, ncol=9, nrow=9)
var2 <- matrix(0, ncol=9, nrow=9)

for(i in 1:9){
  for(j in 1:9){
    output <- optim(par=c(1,1), f.GP, method="SANN", control=list(temp=i, tmax=j))
    obj[i,j] <- output$value
    var1[i,j] <- output$par[1]
    var2[i,j] <- output$par[2]
  }
}
temp <- 1:9; tmax <- 1:9
obj <- as.data.frame(obj)
colnames(obj) <- 1:9
obj <- obj %>% gather('1':'9', key="tmax", value="value")
obj$temp <- rep(1:9,9)
obj %>% ggplot(aes(temp, tmax))+geom_tile(aes(fill=value))
obj[which.min(obj$value),] # tmax=8, temp=1

res3 <- optim(c(1,1), f.GP, method="SANN")
res3
system.time(optim(c(1,1), f.GP, method="SANN"))

res4 <- optim(c(1,1), f.GP, method="SANN", control=list(temp=1, tmax=8))
res4
system.time(optim(c(1,1), f.GP, method="SANN", control=list(temp=1, tmax=8)))

res7 <- optim(res3$par, f.GP, method="SANN")
res7
system.time(optim(res3$par, f.GP, method="SANN"))

res5 <- optim(res4$par, f.GP, method="SANN")
res5
system.time(optim(res4$par, f.GP, method="SANN"))

res6 <- optim(res4$par, f.GP, method="SANN", control=list(temp=1, tmax=8))
res6
system.time(optim(res4$par, f.GP, method="SANN", control=list(temp=1, tmax=8)))

library(GA)
# Function optimisation in one dimension
f <- function(x) (x^2+x)*cos(x)
lbound <- -10; ubound <- 10
curve(f, from=lbound, to=ubound, n=1000, main="f(x)=(x^2+x)*cos(x)")

GA <- ga(type="real-valued", fitness=f, lower=c(th=lbound), upper=ubound)
summary(GA)
plot(GA)

curve(f, from=lbound, to=ubound, n=1000, main="f(x)=(x^2+x)*cos(x)")
points(GA$solution, GA$fitnessValue, col=2, pch=19)

# Function optimisation in two dimension
Rastrigin <- function(x1, x2){
  20 + x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
}
x1 <- x2 <- seq(-5.12, 5.12, by=0.1)

```

```

f <- outer(x1, x2, Rastrigin)
persp3D(x1, x2, f, theta=50, phi=20, col.palette = bl2gr.colors)
filled.contour(x1, x2, f, color.palette=bl2gr.colors)
GA <- ga(type="real-valued",
        fitness = function(x)-Rastrigin(x[1],x[2]),
        lower=c(-5.12, -5.12), upper=c(5.12,5.12),
        popSize=50, maxiter=1000, run=100)
summary(GA)
plot(GA)
filled.contour(x1, x2, f, color.palette = bl2gr.colors,
               plot.axes={axis(1); axis(2);
               points(GA@solution[,1], GA@solution[,2],
                       pch=3, cex=2, col="white",lwd=2)})

# Monitor how GA works
monitor <- function(obj){
  contour(x1, x2, f, drawlabels = F, col=grey(0.5))
  title(paste("iteration=",obj@iter), font.main=1)
  points(obj@population, pch=20, col=2)
  Sys.sleep(0.2)
}

GA <- ga(type="real-valued",
        fitness = function(x)-Rastrigin(x[1],x[2]),
        lower= c(-5.12, -5.12), upper=c(5.12,5.12),
        popSize=50, maxiter=100, monitor = monitor)

# Setting some members of the initial population
suggestedSol <- matrix(c(0.2, 1.5, -1.5, 0.5), nrow=2, ncol=2, byrow=T)
GA1 <- ga(type="real-valued",
        fitness = function(x)-Rastrigin(x[1],x[2]),
        lower=c(-5.12, -5.12), upper=c(5.12, 5.12),
        suggestions=suggestedSol,
        popSize=50, maxiter=1)
head(GA1@population)
GA <- ga(type="real-valued",
        fitness = function(x)-Rastrigin(x[1],x[2]),
        lower=c(-5.12, -5.12), upper=c(5.12, 5.12),
        suggestions=suggestedSol,
        popSize=50, maxiter=100)
summary(GA)

# Constrained optimisation
f <- function(x) 100*(x[1]^2-x[2])^2+(1-x[1])^2
c1 <- function(x) x[1]*x[2] + x[1] - x[2] + 1.5
c2 <- function(x) 10 - x[1]*x[2]

ngrid <- 250
x1 <- seq(0, 1, length= ngrid)
x2 <- seq(0, 13, length= ngrid)
x12 <- expand.grid(x1, x2)
col <- adjustcolor(bl2gr.colors(4)[2:3], alpha=0.2)
plot(x1, x2, type="n", xaxs="i", yaxs="i")
image(x1, x2, matrix(ifelse(apply(x12, 1, c1)<=0, 0, NA), ngrid, ngrid),
      col=col[1], add=T)
image(x1, x2, matrix(ifelse(apply(x12, 1, c2)<=0, 0, NA), ngrid, ngrid),
      col=col[2], add=T)
contour(x1, x2, matrix(apply(x12, 1, f), ngrid, ngrid),
        nlevels=21, add=T)

x <- c(0.8122, 12.3104)

```

```

f(x)
c1(x)
c2(x)
# Matlab solution doesn't meet inequality constraints
# So we're gonna solve it by GA

fitness <- function(x){
  f <- -f(x) # we need to maximise -f(x)
  pen <- sqrt(.Machine$double.xmax) # penalty term
  penalty1 <- max(c1(x), 0)*pen # penalisation for 1st inequality constraint
  penalty2 <- max(c2(x), 0)*pen # penalisation for 2nd inequality constraint
  f - penalty1 - penalty2 # fitness function value
}
GA <- ga("real-valued", fitness=fitness,
        lower=c(0,0), upper=c(1,13),
        maxiter = 1000, run=200, seed=123)
summary(GA)
fitness(GA@solution)
f(GA@solution)
c1(GA@solution)
c2(GA@solution)

plot(x1, x2, type="n", xaxs="i", yaxs="i")
image(x1, x2, matrix(ifelse(apply(x12, 1, c1)<=0, 0, NA), ngrid, ngrid),
      col=col[1], add=T)
image(x1, x2, matrix(ifelse(apply(x12, 1, c2)<=0, 0, NA), ngrid, ngrid),
      col=col[2], add=T)
contour(x1, x2, matrix(apply(x12, 1, f), ngrid, ngrid),
        nlevels=21, add=T)
points(GA@solution[1], GA@solution[2], col="dodgerblue3", pch=3)

# Integer optimisation
AQL <- 0.01; alpha <- 0.05
LTPD <- 0.06; beta <- 0.10
plot(0, 0, type="n", xlim=c(0, 0.2), ylim=c(0,1), bty="l",
     xaxs="i", yaxs="i", ylab="Prob. of acceptance", xlab=expression(p))
lines(c(0,AQL), rep(1-alpha,2), lty=2, col="grey")
lines(rep(AQL,2), c(1-alpha,0), lty=2, col="grey")
lines(c(0,LTPD), rep(beta,2), lty=2, col="grey")
lines(rep(LTPD,2), c(beta,0), lty=2, col="grey")
points(c(AQL, LTPD), c(1-alpha, beta), pch=16)
text(AQL, 1-alpha, labels=expression(paste("(",AQL,"",1-alpha,")")), pos=4)
text(LTPD, beta, labels=expression(paste("(",LTPD,"",beta,")")),pos=4)

decode1 <- function(x){
  x <- gray2binary(x)
  n <- binary2decimal(x[1:11])
  c <- min(n, binary2decimal(x[(11+1):(11+12)]))
  out <- structure(c(n,c), names=c("n","c"))
  return(out)
}

fitness1 <- function(x){
  par <- decode1(x)
  n <- par[1] # sample size
  c <- par[2] # acceptance number
  Pa1 <- pbinom(c, n, AQL)
  Pa2 <- pbinom(c, n, LTPD)
  Loss <- (Pa1-(1-alpha))^2+(Pa2-beta)^2
  -Loss
}

```

```

n <- 2:200 # range of values to search
b1 <- decimal2binary(max(n)) # max number of bits requires
l1 <- length(b1) # length of bits needed for encoding
c <- 0:20 # range of values to search
b2 <- decimal2binary(max(c)) # max number of bits requires
l2 <- length(b2) # length of bits needed for encoding

GA1 <- ga(type="binary", fitness = fitness1,
          nBits = l1+l2, popSize = 100, maxiter = 1000, run=100)
summary(GA1)
decode1(GA1@solution)

plot(0, 0, type="n", xlim=c(0, 0.2), ylim=c(0,1), bty="l",
     xaxs="i", yaxs="i", ylab="P[a]", xlab=expression(p))
lines(c(0,AQL), rep(1-alpha,2), lty=2, col="grey")
lines(rep(AQL,2), c(1-alpha,0), lty=2, col="grey")
lines(c(0,LTPD), rep(beta,2), lty=2, col="grey")
lines(rep(LTPD,2), c(beta,0), lty=2, col="grey")
points(c(AQL, LTPD), c(1-alpha, beta), pch=16)
text(AQL, 1-alpha, labels=expression(paste("(",AQL,"", "1-alpha,"))), pos=4)
text(LTPD, beta, labels=expression(paste("(",LTPD,"", "beta,"))), pos=4)
n <- 87; c <- 2
p <- seq(0, 0.2, by=0.001)
Pa <- pbinom(2, 87, p)
lines(p, Pa, col=2)

decode2 <- function(x){
  n <- floor(x[1]) # sample size
  c <- min(n, floor(x[2])) # acceptance number
  out <- structure(c(n,c), names=c("n", "c"))
  return(out)
}

fitness2 <- function(x){
  x <- decode2(x)
  n <- x[1] # sample size
  c <- x[2] # acceptance number
  Pa1 <- pbinom(c, n, AQL)
  Pa2 <- pbinom(c, n, LTPD)
  Loss <- (Pa1-(1-alpha))^2+(Pa2-beta)^2
  return(-Loss)
}

GA2 <- ga(type="real-valued", fitness=fitness2,
          lower=c(2,0), upper=c(200,20)+1,
          popSize = 100, maxiter=1000, run=100)
summary(GA2)
t(apply(GA2@solution, 1, decode2))

nrep <- 100
systime <- loss <- niter <- matrix(as.double(NA), nrow=nrep, ncol=2,
                                   dimnames=list(NULL, c("Binary", "Real-valued")))

for(i in 1:nrep){
  t <- system.time(GA1 <- ga(type="binary", fitness=fitness1,
                             nBits = l1+l2, monitor=F,
                             popSize = 100, maxiter=1000, run=100))

  systime[i,1] <- t[3]
  loss[i,1] <- -GA1@fitnessValue
  niter[i,1] <- GA1@iter
}

```

```
t <- system.time(GA2 <- ga(type="real-valued", fitness = fitness2,
                           lower=c(2,0), upper=c(200,20)+1,
                           monitor=F, popSize = 100, maxiter=1000, run=100))

systime[i,2] <- t[3]
loss[i,2] <- -GA2@fitnessValue
niter[i,2] <- GA2@iter
}

describe <- function(x) c(Mean=mean(x), sd=sd(x), quantile(x))
t(apply(systime, 2, describe))
t(apply(loss, 2, describe))*1000
t(apply(niter, 2, describe))

# Hybrid GAs
optimArgs = list(method="L-BFGS-B", poptim=0.05, pressel=0.5,
                  control=list(fnscale=-1, maxit=100))
GA <- ga(type="real-valued", fitness=function(x)-Rastrigin(x[1],x[2]),
         lower=c(-5.12, -5.12), upper=c(5.12, 5.12),
         popSize = 50, maxiter=1000, run=100, optim=T)
summary(GA)
plot(GA)
```