Lab 2 NISO Qu 4&5

INSTRUCTIONS TO RUN CODE WITH DATA SET BEERSALES#

Note: Apologies for this. Unfortunately, I could not get the docker working with my file. It was producing strange values although I did my best to get it working. Furthermore, some tests were using 5 seconds which was not long enough for my program to produce results. Also, the function lambda already has functionality in python for lambda functions. lambda is therefore denoted as l. I have the following instructions which allow the program to be run from the bash that work with 'beer_sales' dataset time-series data.

Question 1: Evaluation of expression:

```
"python jtm812.py  -question 1  -expr "(div (exp (ifleq -0.155378496782 1.03118717712
0.206816441971 1.03118717712)) (log 1.09965249175))" -n 10 -x "-0.695851789742
-0.189900702591 -1.31924661442 1.45786585151 -2.31217027969 -0.549620673918
0.250946596997 1.36131035577 -0.96456960303 0.200333135047"
```

Question 2: Evaluation of the fitness function:

```
python jtm812.py  -question 2  -expr "(exp 1.92108563144)" -n 2 -m 1000 -data 'beer_sales'
```

Question 3: Does the GP produce better solutions? (approx 10 min)

```
python jtm812.py -question 3 -n 2 -m 478 -data 'beer_sales' -l 200 -time_budget 600
```

Qu 4 - Pseudocode:

Step 1 - Create an initial population

```
1.Create individual

   1.1 1st value randomly selected (operatorator)
   1.2 2nd and 3rd value is either:
          - random constant selected. is either:
              - special
              - numeric float
              ( weighted by "const_threshold")
          - embedded nest + recursion
          (weighted by "nesting_threshold")
   1.3 individual fitness is calculated
   1.4 if individual fitness is "invalid" or too unfit(very large) is discarded by "birth_fitness_limit"
            try again
   1.5 discard duplicate individuals.
```

        if duplicate: try again
    1.6 returns valid_individual

2.Appends valid individual to population and repeats step 1 until "population_limit" is reached

Step 2 - Tournament selection:

1. elites (fittest n individuals) get automatically selected by "num_elites"
2. 1v1 tournament 2.1 two random individual selected from "num_contestants" 2.2 fittest individual is appended to selected individuals list
3. step 2 is repeated until original population size is restored.

Step 3 - Crossover:

1. elites (n fittest individuals) from selected are added to next_generation before crossover occurs
2. breed population: 2.1. choose two random individuals from selected list 2.2. with recursive exploration build an index of both indivs tree branches (embedded nesting levels) 2.3. randomly select a branch from first individual from index 2.4. randomly select a branch from second individual from index 2.5. replace selected branch in second individual with selected branch from first individual 2.6. calculate fitness 2.7. if not invalid and not too unfit and not duplicate will add to population.
    if not valid:try again. 2.1-2.6
   2.8 id individual valid, add to new generation until original population size is reached.

Step 4 - Mutate population:

1. elites get added to new population and avoid mutation

select a members for mutation by "intra_muation_rate" 2.1 for each member either:
 - skip mutation, and added to new generation

2. - mutation occurs

if individual mutation occurs: 3.1. randomly select an argument in first layer 1, 2, 3
 - if value 1 selected a is swapped for random operator
 - if value 2 or 3:
    randomly either branch mutation or single mutation weighted with "branch_threshold"
    - if branch mutation either:
        - swap constant with a randomly generated new branch
        - swap branch with a random constant
    - if single mutation:
        - if value is nest, traverse deeper to mutate deeper by recursion

```
3.          - if constant swap with another random constant
    3.2. If mutation is invalid or too unfit:
        try again:step 3
    3.3 if mutation is valid, added to new population until original population size is reached
```

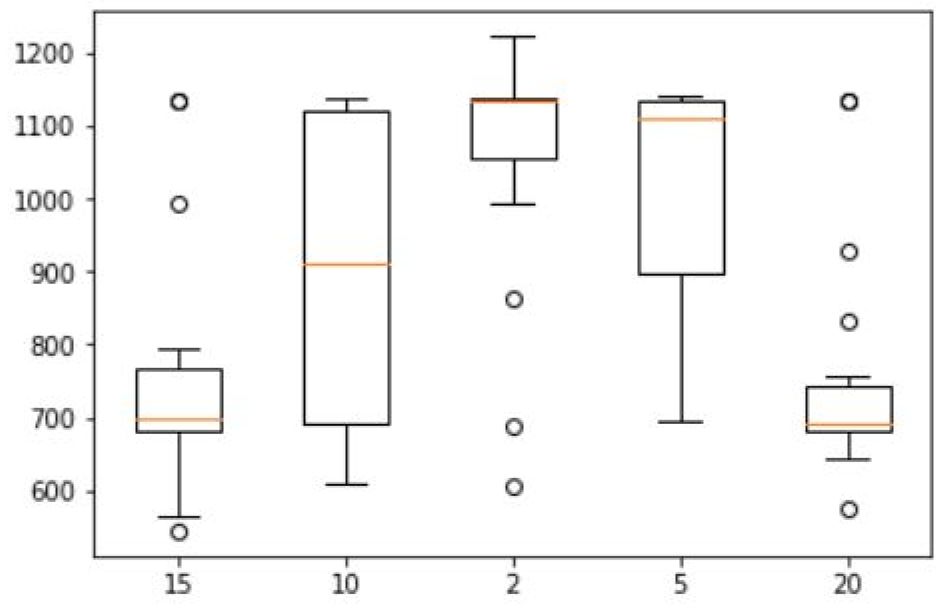
Step 5 - Repeat steps 3-5 for n generations.

Qu 5 - Parameter Experiments

The parameters that I expected to have the biggest influence on the best solution fitness were **population size** and **number of generations**. For each parameter, 100 repetitions were made in total, and the results are shown in a boxplot showing the fitness scores for different parameter values. Other hyperparameters were kept constant.

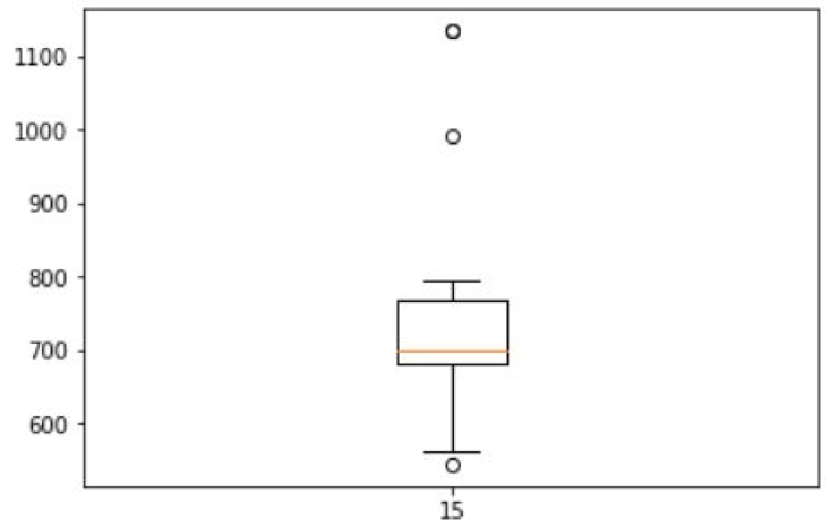
Parameter Generations

Experiments:



num generations

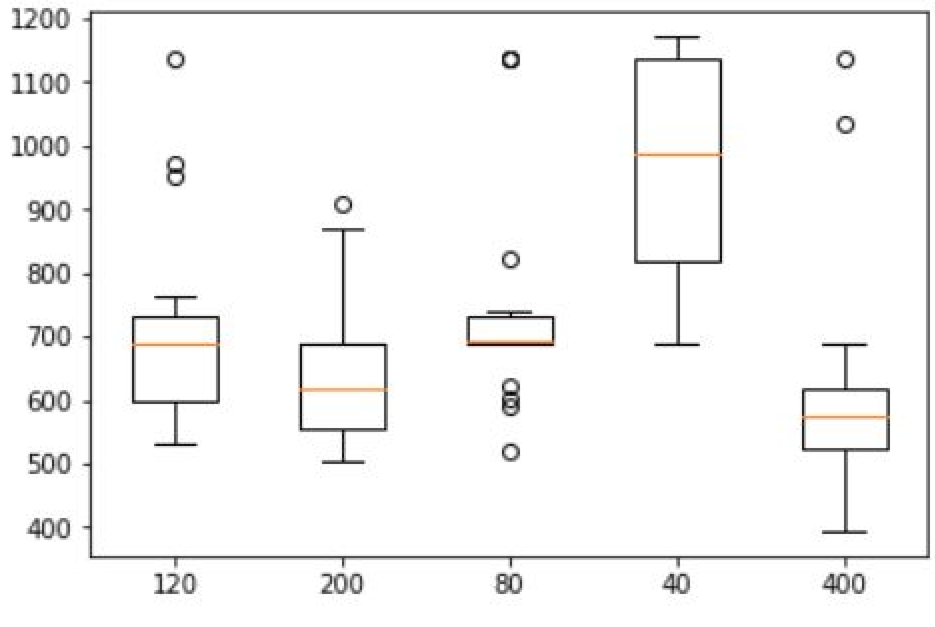
Best overall = **15 generations** (No significant difference at 20):



num generations

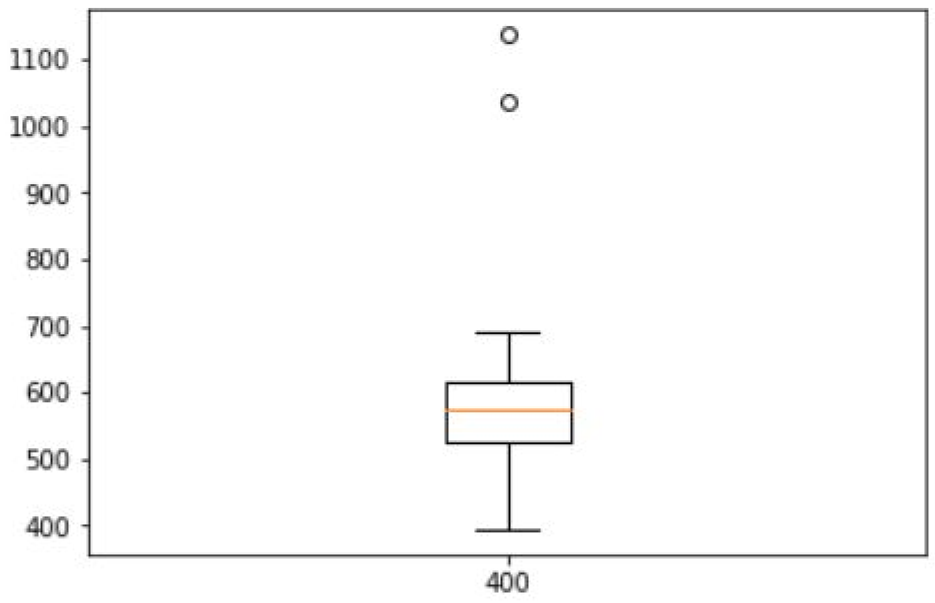
Parameter population size

Experiments:



Population size

Best overall = **400 population size**:


1100
1000
900
800
700
600
500
400
400